VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020 Bc. Lukáš Miško

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

OPTIMALIZACE SMĚROVÁNÍ V PROTOKOLU AD HOC ON-DEMAND DISTANCE VECTOR

AD HOC ON-DEMAND DISTANCE VECTOR ROUTING OPTIMIZATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Lukáš Miško**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Anna Kubánková, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Lukáš Miško ***ID:*** 173708

Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Optimalizace směrování v protokolu Ad hoc On-Demand Distance Vector

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte principy sítí MANET a směrovací protokoly používané v těchto sítích. Zaměřte se na protokol Ad hoc On-Demand Distance Vector (AODV) a podrobně prostudujte jeho principy. Seznamte se s prostředím NS-3. V tomto prostředí navrhněte a vytvořte scénář simulující MANET síť. V navržené síti použijte AODV směrovací protokol. Analyzujte mechanizmus volby peeru v AODV protokolu a prezentujte nedostatky tohoto mechanizmu. Navrhněte a aplikujte nový mechanizmus volby peeru. Otestujte nově navržený mechanizmus v porovnání se standardně používaným.

DOPORUČENÁ LITERATURA:

[1] RFC 3561, „Ad hoc On-Demand Distance Vector (AODV) Routing“, 2003.

[2] VASILIEV D.S., A. ABILOV , V. V. KHRORENKOV. Peer selection algorithm in flying ad-hoc networks. In: Proc. Int. Siberian Conf. Control Commun. (SIBCON). Moskva, 2016, s. 1–4. ISBN 978-1-4673-8383-7.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: Ing. Anna Kubánková, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Táto práca obsahuje teoretický základ pre sietí typu MANET. Zameranie práce je na princípy týchto sietí, ich smerovacích protokolov a to hlave na protokol Ad hoc On-Demand Distance Vector (AODV), jeho implementáciu a následné implementovanie nového mechanizmu voľby peeru. Práca pojednáva o ETX metrike a jej následnej implementácii v AODV protokole. V práci sa nachádzajú simulácie AODV protokolu a následne aj simulácie AODV–ETX protokolu. Simulovanie prebieha v Network Simulator 3. Práca obsahuje porovnanie protokolu AODV a AODV–ETX.

KĽÚČOVÉ SLOVÁ

MANET, AODV, protokol, smerovanie, NS–3, AODV–ETX, ETX metrika, simulácia

ABSTRACT

This thesis contains a theoretical basis for MANET networks. The main focus of the thesis is principles of these networks, their routing protocols and especially on Ad hoc On-Demand Distance Vector (AODV), implementation of this protocol and implementation of new mechanis for peer selection. Thesis contains ETX metric basic and implementation of this metric to AODV protocol. There is a demonstration of simulation of AODV protocol and simulation of AODV–ETX protocol. Simulations are run in Network Simulator 3. AODV and AODV–ETX comparasion are included in thesis.

KEYWORDS

MANET, AODV, protocol, routing, NS–3, AODV–ETX, ETX metric, simulation

MIŠKO, Lukáš. *Optimalizace směrování v protokolu Ad hoc On-Demand Distance Vector*. Brno, 2020, 53 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Anna Kubánková, Ph.D.

VYHLÁSENIE

Vyhlasujem, že svoju diplomovú prácu na tému „Optimalizace směrování v protokolu Ad hoc On-Demand Distance Vector“ som vypracoval samostatne pod vedením vedúceho diplomovej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád bych poděkoval vedoucí diplomové práce pani Ing. Anne KubánkovéPh.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Zoznam obrázkov

Zoznam tabuliek

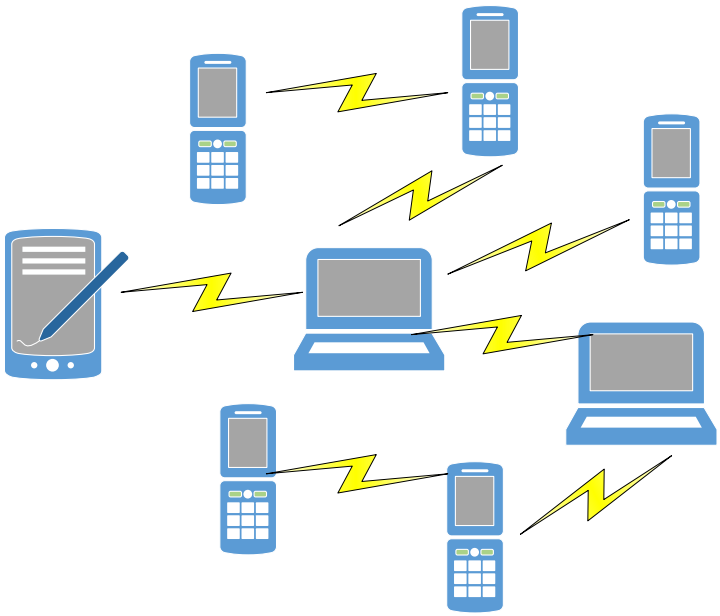
Úvod

Doba v ktorej si deň bez pripojenia k Wi-Fi a internetu ani nevieme predstaviť, sa stáva stále zložitejšou pre uspokojenie užívateľov. Wi-Fi siete sú dostatočné pre mnohých užívateľov, ale nie sú dostatočné mobilné. Takéto siete majú obmedzený dosah a musíme počítať so vždy pripojeným prístupovým bodom. Problematikou mobility sa zaoberajú siete MANET(Mobilná Ad–hoc sieť – Mobile Ad hoc Network). Tieto siete využívajú riešenie bez použitia pevnej infraštruktúry. Aj pri veľkej mobilite uzlov sú tieto siete schopné funkčnosti za pomoci smerovacích protokolov, navrhnutých a zlepšovaných práve za účelom mobility. Táto práca sa bude zaoberať hlavne protokolom reaktívnym a to protokolom AODV(Ad-Hoc On Demand Distance Vector).

Cieľom práce je teoretický rozbor MANET sietí, smerovacích protokolov a ich vlastností. Oboznámenie sa s protokolom AODV a s jeho správami. Práca ďalej pojednáva o voľbe peeru v AODV protokole a upravením tohto mechanizmu za pomoci ETX mechaniky. V práci budú vytvárané simulácia za pomoci nástroja NS–3(Network Simulator 3), ktorý bude bližšie popísaný v jednej časti práce. Simulácie budú vytvárané s protokolom AODV a následne aj s upraveným protokolom AODV–ETX. Výsledky simulácii sú zhodnotené v poslednej kapitole tejto práce.

1 MANET sieť

Sieť s pomenovaním Ad hoc je taká, v ktorej komunikujú zariadenia ako sú napríklad počítače, mobily medzi sebou za pomoci bezdrôtových zariadení. MANET(Mobilná Ad–hoc sieť – Mobile Ad hoc Network) je narozdiel od Ad hoc siete autonómny systém, neštrukturovaná sieť, ktorá podporuje viac bodovú komunikáciu prostredníctvom IP smerovania bez pevných prístupových bodov a akejkoľvek centralizovanej forme správy [1]. V sieti MANET sa uzly (laptopy, mobilné telefóny, ostatné bezdrôtové zariadenia) môžu pohybovať voľne a úplne náhodne keďže sa topológia často mení. Každý uzol sa môže chovať ako klient a prijímať informácie alebo ako server, ktorý odosiela poskytnuté dáta cez sieť. Štruktúra MANET siete je zobrazená na obr.1.1



Obr. 1.1: Mobilné stanice v sieti MANET

Pre funkčnosť siete, nie všetky uzly musia byť v dosahu všetkých ostatných uzlov. Ak stanica bude mať rádiový dosah aspoň na jeden z prvkov v sieti, stáva sa plnohodnotným členom tejto siete. Takáto sieť môže fungovať ako samostatná alebo môže byť súčasťou internetu.
Hlavnou výzvou siete MANET je nepretržitá správa informácii, každého zariadenia, aby bolo zaistené správne smerovanie prevádzky. MANET pozostáva z peer-to-peer spojenia, sieť je samo opravujúca sa a zároveň samostatne formovaná. Sieť môže

byť využívaná v oblasti bezpečnosti cestnej premávky, pri senzoroch v domácnosti, zdravotníctve, v robotike atď... [2]

1.1 Vlastnosti MANET

Hlavné vlastnosti MANET, ktoré sú pre ňu charakteristické: úplná absencia centralizovanej kontroly, nedostatočné prepojenie medzi uzlami, rýchla mobilita uzlov, dynamicky sa meniaca topológia siete, vysielací rádiový kanál je zdieľaný, fyzická zraniteľnosť, nezabezpečené pracovné prostredie a obmedzená dostupnosť zdrojov, ako sú kapacita CPU(Centrálna Procesorová Jednotka – Central Processing Unit), pamäťový výkon, šírka pásma a výkon batérie.[3]

- Dynamická topológia siete: V sieti sa uzly pohybujú náhodne a v rôznych smeroch. Sieťová topológia sa neustále mení, náhodne, nepredvídateľne a v rôznom čase. Systém sa takýmto zmenám musí prispôsobovať.
- Nízka šírka pásma: Siete tohto typu majú menšiu kapacitu a kratší dosah ako siete kde je infraštruktúra pevná. Priepustnosť bezdrôtovej komunikácie je nižšia, pre nevýhodné vlastnosti ako šum, rušenie, ktoré je možné odstrániť len do určitej miery.
- Obmedzený výkon batérie: Väčšina uzlov v infraštruktúre majú menšiu kapacitu batérie a preto úspora energie je najdôležitejším kritériom dizajnu siete.
- Decentralizovaná kontrola: V dôsledku nespoľahlivých spojení, dynamickej topológie, fungovanie systému závisí na spolupráci zúčastnených uzlov.
- Nespoľahlivá komunikácia: Povaha zdieľaného média a nestabilita kvality kanálov bezdrôtových liniek môže viesť k vysokej miere straty paketov a nestabilite spojenia. Toto je bežný jav, ktorý vedie k poklesu priepustnosti.
- Slabá ochrana: Mobilné uzly sú kompaktné, mäkké a ručne držané. Ako doba postupuje dopredu tak sa prenosné zariadenia zmenšujú a tak sa stávajú viac náchylné na zničenie, stratenie alebo odcudzenie. Tak isto sú náchylnejšie na útoky ako je napríklad rušenie, odposluch, podsúvanie škodlivých dát.
- Škálovateľnosť: Keď zoberieme do úvahy obmedzený výkon a pamäť mobilných zariadení, škálovateľnosť sa stáva jedným z kľúčových problémov pri tvorbe veľkých sietí o 10000 a viac uzloch. [3][4]

1.2 Smerovanie

Smerovanie znamená zvolenie si cesty k cieľu. V MANET sieti je veľmi dôležitou úlohou zvoliť správnu cestu od zdroja ku cieľu. Smerovanie sa používa v internetovej

sieti, v mobilných technológiách alebo aj v elektronických dátových sieťach. Na smerovanie používame protokoly obsahujúce súbory pravidiel, vďaka ktorým môžu dve alebo viac zariadený navzájom komunikovať. V mobilných sieťach sa smerovanie vykonáva najčastejšie pomocou smerovacích tabuliek, ktoré sa uchovávajú v cash pamäti týchto uzlov.[5] Používajú sa dva základné typy smerovania:

- Statické smerovanie: Nastavované permanentne administrátorom siete a tak pri behu siete nie sú v smerovacej tabuľke aktívne vykonávané zmeny.
- Dynamické smerovanie: Je vykonávané automaticky, reaguje na zmeny v sieti, smerovače si vytvárajú svoje smerovacie tabuľky sami za pomoci smerovacích protokolov. Dynamické smerovanie je flexibilnejšie oproti statickému.

1.3 Smerovacie protokoly

Smerovacie protokoly delíme na dva základné typy:

1. Distance vector
2. Link state

1.3.1 Distance vector

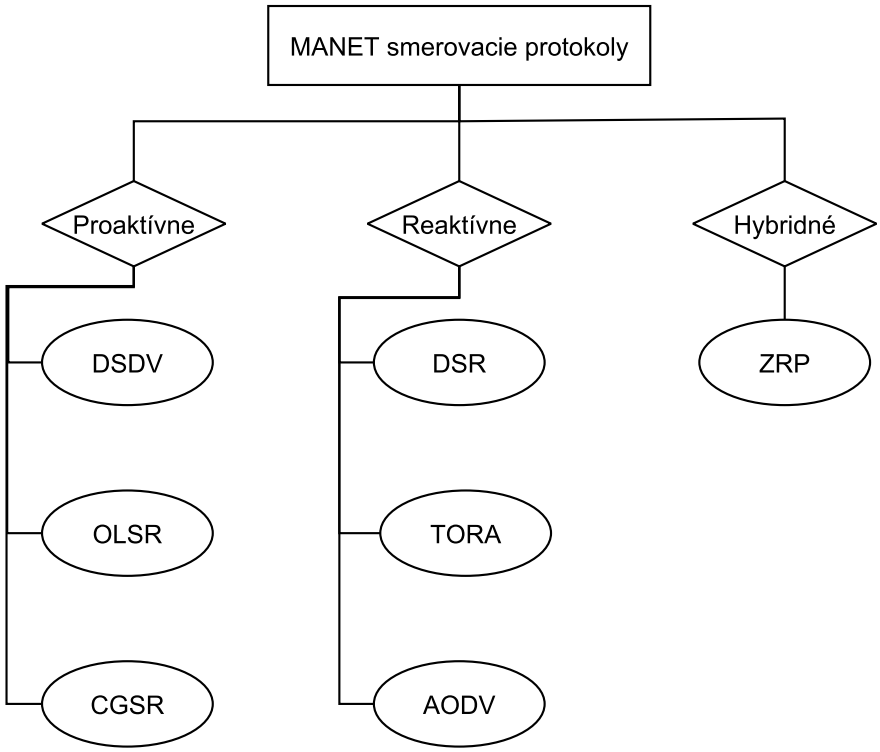
Tento typ protokolov používa vzdialenosť ako hlavnú veličinu na určenie najlepšej cesty k cieľu. Distance vector protokoly zvyčajne odosielajú smerovaciu tabuľku svojím susedom, teda priamo pripojeným smerovačom, ktorý používa rovnaký smerovací protokol.
V porovnaní s link state smerovacími protokolmi sú ľahšie konfigurovateľné, vyžadujú menšiu správu, ale sú citlivé na smerovacie slučky.

1.3.2 Link state

Link state protokoly majú rovnaký účel ako Distance vector a to nájsť najlepšiu trasu od zdroja k cieľu. Narozdiel od Distance vector neposielajú celú smerovaciu tabuľky, namiesto toho reagujú len na zmeny v topológii siete. Vyžadujú väčšie využitie procesora a pamäte smerovača.[6]

2 Smerovacie protokoly v MANET

Existuje niekoľko druhov smerovacích protokolov pre bezdrôtové Ad hoc siete. Tieto smerovacie protokoly sa kategorizujú do dvoch hlavných skupín na proaktívne a reaktívne. Smerovacie protokoly, ktoré využívajú vlastností proaktívnych aj reaktívnych sa nazývajú hybridné.[5] Rozdelenie smerovacích protokolov je zobrazené na obr. 2.1 a porovnanie vlastností protokolov v tabuľkách 2.1, 2.2, 2.3.



Obr. 2.1: Rozdelenie smerovacích protokol v MANET

2.1 Proaktívne (*Proactive*) smerovacie protokoly

Proaktívne smerovacie protokoly alebo aj protokoly riadené tabuľkou sú si podobné s protokolmi pre siete pripojené káblom a prichádzajú ako ich prirodzené rozšírenie. Pri proaktívnom smerovaní má každý uzol jednu alebo viac tabuliek, ktoré obsahujú najnovšie informácie o trasách k akémukoľvek uzlu v sieti. Každý riadok obsahuje ďalší skok (*hop*) pre dosiahnutie uzlu alebo podsiete a cenu (*cost*) tejto cesty. Proaktívne protokoly sa medzi sebou líšia v tom ako poskytujú informácie o zmenách

topológie siete cez ostatné uzly. V tomto type protokolov si každý uzol udržuje aktuálne smerovacie informácie o všetkých uzloch v sieti.
Každý uzol tu udržuje smerovaciu tabuľku a za pomoci broadcastu ju posiela na ostatné uzly ako náhle dôjde k zmene v sieťovej topológii. Keď uzol ako zdroj správy potrebuje odoslať informácie pre iný uzol a získať trasu k nemu, nazrie do svojej smerovacej tabuľky a vyberie si trasu. Smerovacia tabuľka sa v sieti pravidelne vymieňa aj keď nenastane žiadna zmena v topológii. Každý zúčastnený uzol má informácie o smerovaní všetkých uzlov v sieti, aj keď väčšina týchto informácii je preň nežiadúca.[7]

- Výhoda: Pri komunikácii za pomoci proaktívnych protokolov dochádza k minimálnemu oneskoreniu a trasy sú vždy aktuálne.
- Nevýhoda: Trasy bývajú často prerušené v dôsledku mobility uzlov.

2.1.1 DSDV

DSDV(Destination Sequenced Distance Vector) smerovací protokol, závisí od použitia tradičného Bellman–Ford smerovacieho algoritmu so špecifickými vylepšeniami. Tento protokol bol vyvinutý C. Perkins a P.Bhagwat v roku 1994.[8] Každý uzol udržiava smerovaciu tabuľku, ktorá obsahuje všetky dostupné uzly, množstvo skokov do jednotlivých cieľov a sekvenčné čísla cieľových uzlov. Sekvenčné čísla sa využívajú na rozdelenie zastaralých a nových trás. Uzly občasne posielajú menšie aktualizácie smerovacie tabuľky svojim susedom a naopak celé smerovacie tabuľky zasielajú ak v sieti nastane veľká zmena. Tento protokol je vhodný pre siete s malým počtom uzlov. Keďže neexistuje žiadny formálny opis algoritmu tohto protokolu, nevyskytuje sa v žiadnych komerčných implementáciách.[8][9]

2.1.2 OSLR

OSLR(Optimized Link State Routing) protokol preberá stabilitu z link state algoritmov a jeho výhoda spočíva v okamžitom prístupe k smerovacej trase, pretože je protokolom proaktívnym. OSLR je teda optimalizovaný link state protokol pre mobilné Ad hoc siete. Protokol minimalizuje veľkosť zo zaplavenia riadiacim prenosom pomocou vybraných uzlov, nazývaných MPR(Multipoint relays), používaných na zasielanie riadiacich správ. Táto technika výrazne znižuje počet opakovaných prenosov potrebných na zaplavenie správy do všetkých uzlov siete. OSLR stále udržuje trasy ku všetkým cieľom v sieti, protokol je prínosný pre režim prevádzky kde komunikuje veľké množstvo uzlov s ďalším veľkým množstvom uzlov a taktiež tam kde sa zdrojové a cieľové uzly časom menia. Je zvlášť vhodný pre veľké a husté siete, kde zaisťuje väčšiu optimalizáciu v porovnaní s klasickými link state protokolmi.[10]
Protokol OSLR:

- Je navrhnutý aby pracoval distribuovaným spôsobom bez centrálnej entity
- Nevyžaduje spoľahlivý prenos riadiacich správ.
- Nevyžaduje postupné doručovanie správ, každá správa obsahuje poradové číslo. Príjemca kontrolnej správy môže v prípade potreby ľahko zistiť, ktoré informácie sú novšie aj keď boli počas prenosu správy preposlané.
- Poskytuje podporu pre rozšírenie ako napríklad režim spánku, smerovanie multicastu a podobné.

2.1.3 CGSR

CGSR(Clusterhead Gateway Switch Routing) používa základy DSDV algoritmu popisovanom v sekcii 2.1.1. Mobilné uzly sa spájajú do skupín (*cluster*) a následne je zvolená hlava skupiny. Všetky uzly, ktoré sú v komunikačnom dosahu hlavy patria do jej skupiny. Bránou sa stáva uzol, ktorý je v komunikačnom dosahu dvoch alebo viacerých hláv skupiny. Zhoršovanie výkonu môže spôsobiť časté volenie hlavy skupiny a preto CGSR používa algoritmu LCC(Least Cluster Change). V tomto algoritme dochádza k zmene hlavy skupiny iba vtedy, ak zmena v sieti spôsobí prípad, že sa dve hlavy stretnú v jednej skupine alebo ak sa jeden uzol presunie z dosahu všetkých hláv skupiny.[11] Všeobecne algoritmus pracuje nasledujúcim spôsobom:

1. Uzol, ktorý je zdrojom správy prenáša paket k hlave skupiny.
2. Hlava skupiny prenáša paket doručený od zdroja ku bráne, ktorá je prepojená s hlavou skupiny zdroja a tak isto s hlavou skupiny, ktorá je cestou k cieľu.
3. Brána paket odošle k hlave skupiny a opakujú sa body 1. a 2. až kým paket nedorazí k skupine cieľa.
4. Cieľová hlava skupiny následne paket odošle cieľu.

2.2 Reaktívne (*On–Demand*) smerovacie protokoly

Reaktívne smerovacie protokoly, ináč nazývané aj protokoly na vyžiadanie alebo protokoly riadené dopytom zdroja sa nazývajú tak, pretože trasa sa objaví iba vtedy, keď ju zdroj vysielania informácie potrebuje. Využívajú „lenivý“ prístup k smerovaniu. Nezachovávajú ani neustále neudržujú ich smerovacie tabuľky aktuálne s najnovšími poznatkami o topológii trasy.
Ak chce zdroj komunikovať s cieľom, musí vyvolať mechanizmus zisťovania trasy aby našiel cestu k svojmu cieľu. Proces zisťovanie trasy sa dokončí vtedy ako náhle nájde k cieľu trasu alebo sa identifikujú všetky možné cesty. Ako náhle sa trasa medzi zdrojom a cieľom vytvorí, je vďaka mechanizmu údržby trasy stále udržiavaná až kým cieľ nebude neprístupný alebo trasa už od zdroja nebude požadovaná.[7]

- Výhoda: Protokoly posielajú čo najmenej servisných správ a tak šetria šírkou pásma, vhodné použitie pri malých sieťach s vysokou mobilitou uzlov.
- Nevýhoda: Tieto algoritmy ponúkajú vysokú latenciu pri prehľadávaní siete.

2.2.1 DSR

DSR(Dynamic Source Routing protocol) je jednoduchý a efektívny protokol navrhnutý špeciálne pre použitie vo viac skokových (*multi–hop*) bezdrôtových Ad hoc sieťach tvorených mobilnými uzlami. Pri využívaní protokolu DSR je sieť kompletne samo organizujúca a samo nastavujúca sa, nepotrebuje žiadnu existujúcu infraštruktúru alebo administratívu. Ako sa uzly do siete pripájajú a odpájajú, poprípade ako sa mení bezdrôtová komunikácia zdroja a cieľa, všetko smerovanie je automaticky rozhodované a udržované vďaka DSR protokolu. Tento protokol poskytuje reaktívne služby vo vysokom štandarde aby tak mohol zaistiť úspešné doručenie dát paketov v prípade, že sa uzol hýbe alebo nastanú v sieti zmeny. Dva hlavné mechanizmy, ktoré obsahuje DSR protokol a ktoré spolu úzko spolupracujú, sú hlavným kľúčom k tomu aby mohol zaisťovať služby nachádzania a údržby Ad hoc siete:

1. Nájdenie trasy (*Route Discovery*): je to mechanizmus s ktorým uzol **O** (odosielateľ), ktorý chce poslať paket do cieľa uzlu **P** (prijímateľ), zistí trasu k P. Tento mechanizmus je použitý len v prípade keď sa **O** snaží poslať paket do **P**, ale ešte nepozná trasu k **P**.
2. Údržba trasy (*Route Maintenance*): je mechanizmus vďaka ktorému je **O** zistiť, či počas toho ako používa trasu k **P** nenastala zmena v sieťovej topológii, ktorá by mala za následok prerušenie trasy k **P**. Keď tento mechanizmus indikuje prerušenie zdrojovej trasy, **O** môže skúsiť použiť inú trasu o ktorej vie, že vedie k **P** alebo môže využiť mechanizmus nájdenia trasy znova a tak nájsť novú trasu na posielanie dátových paketov pre **P**. Mechanizmus údržby trasy sa využíva len v prípade že **O** práve posiela pakety do **P**.[12]

V DSR tieto dva mechanizmy pracujú výlučne na „požiadavku“ (*„On–Demand“*). Vo všeobecnosti narozdiel od ostatných protokolov DSR nevyžaduje žiadny druh periodických paketov.

2.2.2 TORA

TORA(Temporally Ordered Routing Algorithm) protokol je vysoko adaptívny, efektívny a škálovateľný distribuovaný smerovací algoritmus založený na koncepte obrátenia linky.[11] Protokol je navrhnutý pre veľmi dynamické mobilné, viac skokové (*multi–hop*) bezdrôtové siete. Tento smerovací protokol patrí do skupiny reaktívnych a teda je inicializovaný zdrojom (*On–Demand*). Protokol nájde viac trás zo zdroja

k cieľovému uzlu. Hlavná funkcia protokolu TORA je tá, že riadiace správy sú lokalizované do veľmi malých skupín uzlov blízko výskytu zmeny v topológii. Aby táto funkcia mohla byť zabezpečená musia uzly uchovávať smerovacie informácie o susedných uzloch.[11] Tento protokol má tri základné funkcie:

- Vytvorenie trasy (*Route creation*)
- Údržba trasy (*Route maintenance*)
- Vymazanie trasy (*Route erasure*)

2.2.3 AODV

AODV(Ad-Hoc On Demand Distance Vector) je vylepšením algoritmu DSDV preberanom v 2.1.1. Protokol minimalizuje množstvo broadcastu tým, že trasy vytvára na požiadanie, opozitom je k tomu protokol DSDV, ktorý udržuje list všetkých trás. Aby uzol našiel cestu k destinácii pošle za pomoci boradcastu paket požiadavky trasy (*route request packet*). Susedné uzly posielajú broadcastom paket ďalším susedom, tí ďalším až sa paket nedostane k uzlu, ktorý má informácie o trase k destinácii alebo kým paket požiadavky trasy nedorazí až k cieľu. Uzol zahadzuje pakety požiadavky trasy, ktoré už videl. Tieto pakety používajú sekvenčné číslo aby zaistili trasy bez slučiek a aby sa zaistilo, že uzol, ktorý odpovie na požiadavku odpovedal len s najaktuálnejšími informáciami. Tento protokol používa len symetrické linky.[11]

2.3 Hybridný smerovací protokol

Hybridný smerovací protokol v zásade kombinuje výhody protokolov proaktívnych a reaktívnych. Tieto protokoly sú svojou povahou adaptívne a prispôsobujú sa podľa zón a tiež podľa polohy zdrojových a cieľových uzlov. Jedným z najpopulárnejších hybridných protokolov je ZRP(Zone Routing Protocol).[13]

2.3.1 ZRP

Pri smerovacom protokole ZRP je celá sieť rozdelená do rôznych zón a následne je sledovaná poloha zdrojového a cieľového mobilného uzlu. AK je zdrojový aj cieľový uzol prítomný v tej istej zóne, využije sa na prenos dátových paketov medzi nimi proaktívne smerovanie. V opačnom prípade ak je zdrojový uzol v jednej zóne a cieľový uzol v inej, využíva sa na prenos dátových paketov reaktívne smerovanie. [13]

Vlastnosti	***Proaktívne***
Štruktúra smerovania	Plochá aj hierarchická
Získanie trasy	Smerovacie tabuľky
Zaťaženie smerovania	Veľké
Oneskorenie	Malé vďaka smerovacím tabuľkám
Škálovateľnosť	>100 uzlov, malá
Informácie o smerovaní	Vždy prístupné
Periodické aktualizácie	Vždy keď sa zmení topológia
Mobilita	Periodické aktualizácie
Požiadavky na pamäť	Vysoké
Požiadavky na šírku pásma	Vysoké
Požiadavky na energiu	Vysoké

Tab. 2.1: Vlastnosti proakívnych smerovacích protokolov

Vlastnosti	***Raktívne***
Štruktúra smerovania	Plochá
Získanie trasy	Na vyžiadanie
Zaťaženie smerovania	Malé
Oneskorenie	Veľké
Škálovateľnosť	Pre malé siete do 100 uzlov
Informácie o smerovaní	K dispozícii keď je potreba
Periodické aktualizácie	Nie sú potrebné
Mobilita	Údržba trasy
Požiadavky na pamäť	Nízke
Požiadavky na šírku pásma	Nízke
Požiadavky na energiu	Nízke

Tab. 2.2: Vlastnosti reaktívnych smerovacích protokolov

Vlastnosti	***Hybridné***
Štruktúra smerovania	Hierarchická
Získanie trasy	Kombinácia oboch
Zaťaženie smerovania	Stredné
Oneskorenie	Vo vnútri zón malé
Škálovateľnosť	>1000 uzlov, vytvorené pre veľké siete
Informácie o smerovaní	Kombinácia oboch
Periodické aktualizácie	Áno
Mobilita	Kombinácia oboch
Požiadavky na pamäť	Stredné
Požiadavky na šírku pásma	Stredné
Požiadavky na energiu	Stredné

Tab. 2.3: Vlastnosti hybridných smerovacích protokolov

3 Rozbor protokolu AODV

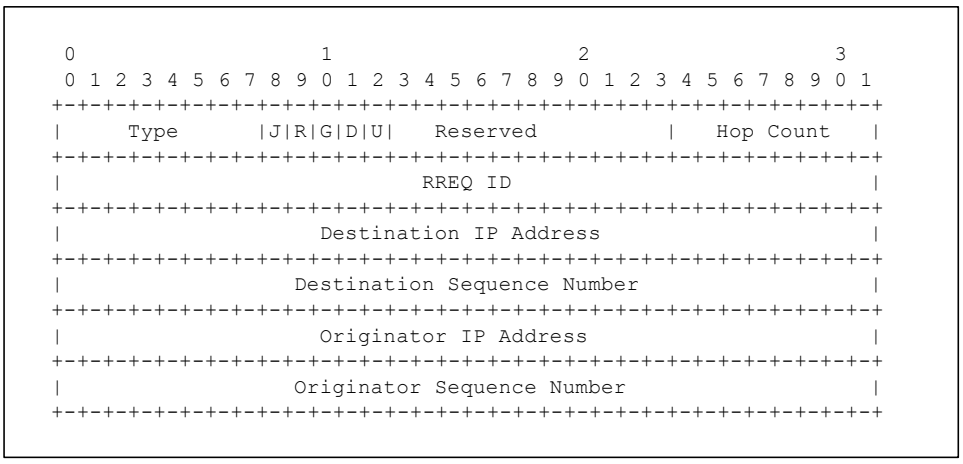
AODV algoritmus umožňuje dynamické, samo spúšťacie, viac skokové smerovanie medzi zúčastnenými mobilnými uzlami, ktoré si prajú vytvoriť a udržovať Ad hoc sieť. Umožňuje mobilným uzlom získať smerovaciu trasu pre nové ciele rýchlo a nevyžaduje od uzlov aby si udržiavali smerovacie trasy k cieľom ak ich práve nepotrebujú pre komunikáciu. Použitie AODV zaisťuje aby sa v sieti nevytvárali zbytočné slučky a tak isto sa vďaka nemu vyhneme javu „počítania do nekonečne“ spojeným s Bellman–Ford algoritmom.[14] Protokol definuje správy typu RREQ(Požiadavka na trasu – Route Request), RREP(Odpoveď na trasu – Route Reply), RERR(Chyba na trase – Route Error). Tieto typy správ sú prijímané prostredníctvom UDP(Používateľský Datagramový Protokol – User Datagram Protocol) a využívajú IP hlavičku. Protokol teda očakáva, že odosielateľ správy použije svoje IP adresu ako IP adresu pôvodcu správy. Na broadcastové správy sa používa IP adresa(*255.255.255.255*).

3.1 Správy AODV

Ako je spomínané v kapitole 3 protokol AODV definuje tri typy správ, ktoré budú popísané v tejto sekcii.

3.1.1 RREQ

Keď sa chce odosielateľ, teda zdrojový uzol, spojiť s cieľovým uzlom a nepozná žiadnu cestu k tomuto uzlu, vyšle riadiaci paket RREQ ako broadcast na všetky okolité uzly. ID požiadavku sa zvyšuje zakaždým, keď zdrojový uzol pošle nový RREQ. Ako RREQ postupuje od uzlu do ďalšieho uzlu vytvára automaticky reverzné cesty zo všetkých zúčastnených uzlov späť k zdroju RREQ správy. Tento proces je tiež nazývaný ako RPS(Nastavenie spätnej cesty – Reverse Path Setup). Štruktúra správy RREQ je zobrazená na obr.3.1.[14][15]



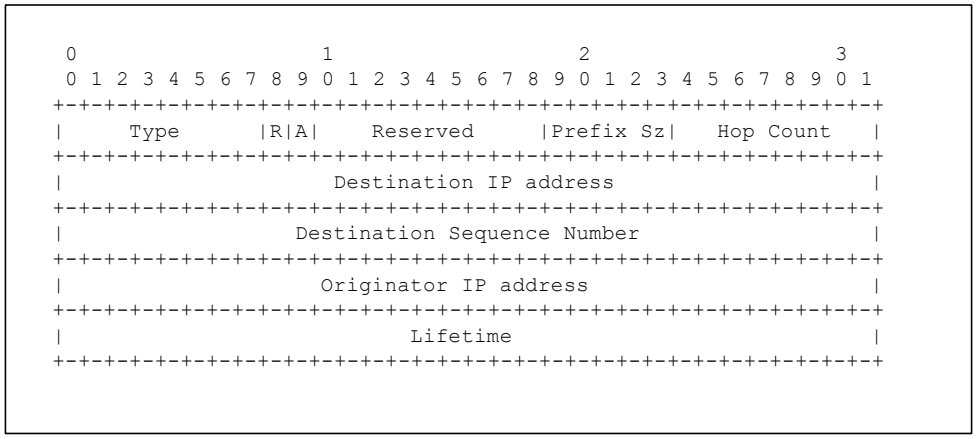
Obr. 3.1: Štruktúra správy RREQ

Formát správy RREQ zobrazenej na obrázku vyššie pozostáva z nasledujúcich polí:

- **Type**: nastavené na 1.
- **J**(Join flag): príznak pre pripojenie; rezervované pre multicast.
- **R**(Repair flag): príznak pre opravu; rezervované pre multicast.
- **G**(Gratuitous RREP flag): príznak zbytočnosti; indikuje či má byť správa poslaná za pomoci unicastu uzlu, ktorý je špecifikovaný v poli Destination IP Address.
- **D**(Destination only flag): príznak cieľa; uvádza či na správu RREQ môže reagovať len cieľový uzol.
- **U**(Unknown sequence number): príznak neznámeho sekvenčného čísla; poukazuje na to, že sekvenčné číslo je neznáme.
- **Reserved**: nastavené na hodnotu 0. Pri prijatí je ignorované.
- **Hop Count**: počet skokov do pôvodného uzlu, ktorý poslal správu až k uzlu, ktorý túto správu spracúva.
- **RREQ ID**: sekvenčné číslo, ktoré jednoznačne identifikuje RREQ spolu v spojení s IP adresou pôvodcu správy.
- **Destination IP Address**: IP adresa cieľového uzlu.
- **Destination Sequence Number**: posledné sekvenčné číslo prijaté od pôvodcu správy pre akúkoľvek cestu k cieľu.
- **Originator IP Address**: IP adresa uzlu, ktorý vytvoril RREQ požiadavku.
- **Originator Sequence Number**: aktuálne sekvenčné číslo, ktoré bude použité na vstupnom bode trasy k pôvodcovi RREQ žiadosti.[14]

3.1.2 RREP

Ak je uzol cieľom alebo má platnú trasu k cieľu, odošle správu RREP naspäť ku zdroju. Uzol pri prijímaní správy RREQ od susedného uzlu zaznamená aj jeho adresu. Keď sa nakoniec nájde cieľový uzol, správa RREP bude vysielaná naspäť po trase, po ktorej bola posielaná správa RREQ a tak nebude za potreby ďalšie broadcastové vysielanie. Správa RREP sa vracia naspäť k zdroju na základe spätnej cesty, ktorá je zaznamenaná. Štruktúra správy RREP je zobrazená na obr.3.2.[14][15]



Obr. 3.2: Štruktúra správy RREP

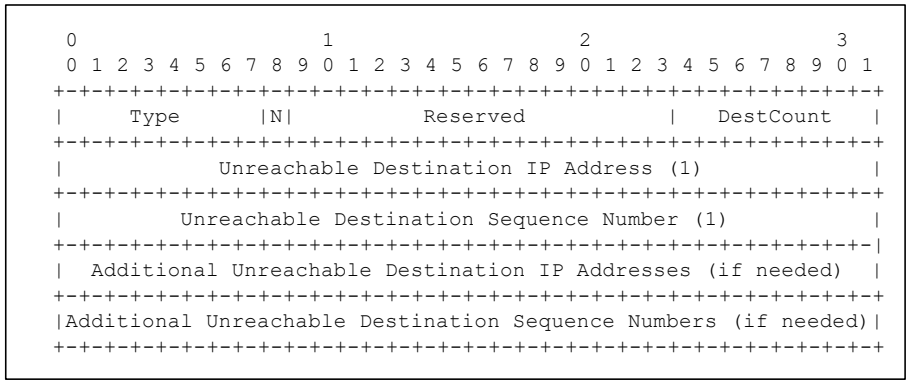
Formát správy RREP zobrazenej na obrázku vyššie pozostáva z nasledujúcich polí:

- **Type**: nastavené na 2.
- **R**(Repair flag): príznak pre opravu; rezervované pre multicast.
- **A**(Acknowledgment required): príznak pre požiadavku potvrdenia. Tento bit sa používa v prípade, že spojenie cez ktoré je správa RREP odoslaná môže byť nespoľahlivé alebo jednosmerné. Ak správa RREP obsahuje **A** bit, očakáva sa, že príjmateľ správy RREP vratí správu RREP–ACK(Potvrdenie odpovede na trasu – Route Reply Acknowledge) popisovanú v sekcii 3.1.4.
- **Reserved**: nastavené na hodnotu 0. Pri prijatí je ignorované.
- **Prexif size**: ak je toto pole nenulové, 5–bitová veľkosť prefixu určuje, či ďalší uvedený skok môže byť použitý pre viac uzlov s rovnakým smerovacím prefixom ako požadovaný cieľ.
- **Hop Count**: počet skokov do pôvodného uzlu, ktorý poslal správu až k uzlu, ktorý túto správu spracúva.
- **Destination IP Address**: IP adresa cieľového uzlu.

- **Destination Sequence Number**: sekvenčné číslo, ktoré je spojené so smerovaním.
- **Originator IP Address**: IP adresa uzlu, ktorý odosiela RREP správu ako odpoveď na požiadavku RREQ.
- **Lifetime**: čas v milisekundách počas ktorého uzly, ktoré obdržali RREP správu považujú cestu za stále platnú.[14]

3.1.3 RERR

Každý uzol monitoruje jeho susedné uzly. Ak je trasa prerušená alebo neplatná, odošle sa notifikácia v podobe RERR správy, ktorá upozorní uzly, ktoré používajú túto trasu na jej neplatnosť. Táto správa sa generuje aby sa zabránilo opakovanému prenosu neplatnou trasou. Štruktúra správy RREP je zobrazená na obr.3.3.[14][15]



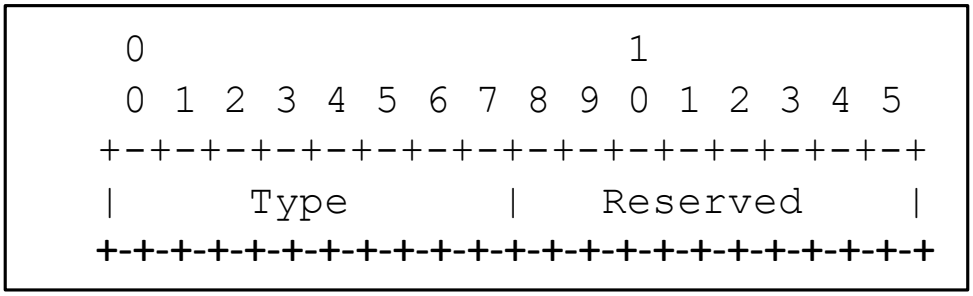
Obr. 3.3: Štruktúra správy RRER

Formát správy RERR zobrazenej na obrázku vyššie pozostáva z nasledujúcich polí:

- **Type**: nastavené na 3.
- **N**(No delete flag): príznak pre neodstránenie; nastaví sa keď uzol previedol lokálne opravy linky a ostatné uzly by nemali túto cestu vymazať.
- **Reserved**: nastavené na hodnotu 0. Pri prijatí je ignorované.
- **DesCount**: počet nedostupných destinácií, ktoré správa obsahuje. Toto číslo musí byť nastavené aspoň na hodnotu 1.
- **Unreachable Destination IP Address**: IP adresa cieľového nedostupného uzlu.
- **Unreachable Destination Sequence Number**: sekvenčné číslo, spojené s nedostupným uzlom.[14]

3.1.4 RREP–ACK

Táto správa sa posiela odosielateľovi ako odpoveď na správu RREP, ktorá mala nastavený príznak **A**. Správy takéhoto typu sa obvykle vytvárajú vtedy keď hrozí nebezpečenstvo jednosmerných liniek, ktoré by mohli mať za následok bránenie dokončeniu procesu zisťovania trasy. Štruktúra správy RREP–ACK je zobrazená na obr.3.4.[14][15]



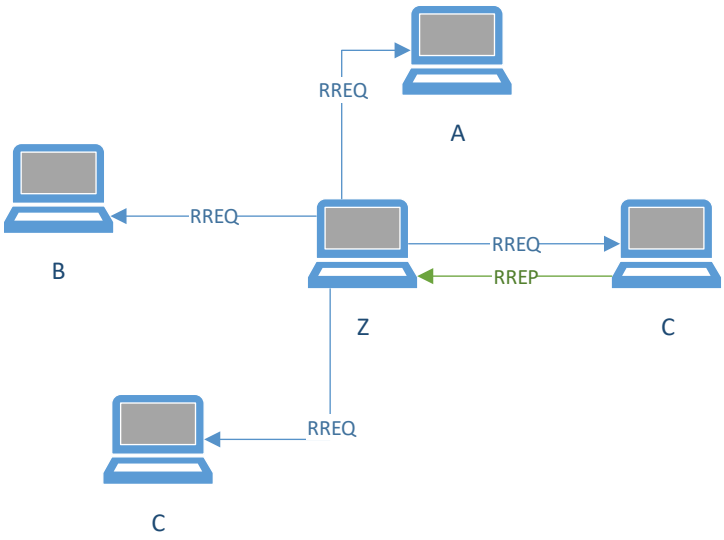
Obr. 3.4: Štruktúra správy RREP–ACK

Formát správy RREP–ACK zobrazenej na obrázku vyššie pozostáva z nasledujúcich polí:

- **Type**: nastavené na 3.
- **Reserved**: nastavené na hodnotu 0. Pri prijatí je ignorované.[14]

3.1.5 Proces objavovania trasy *(Router Discovery Process)* v AODV

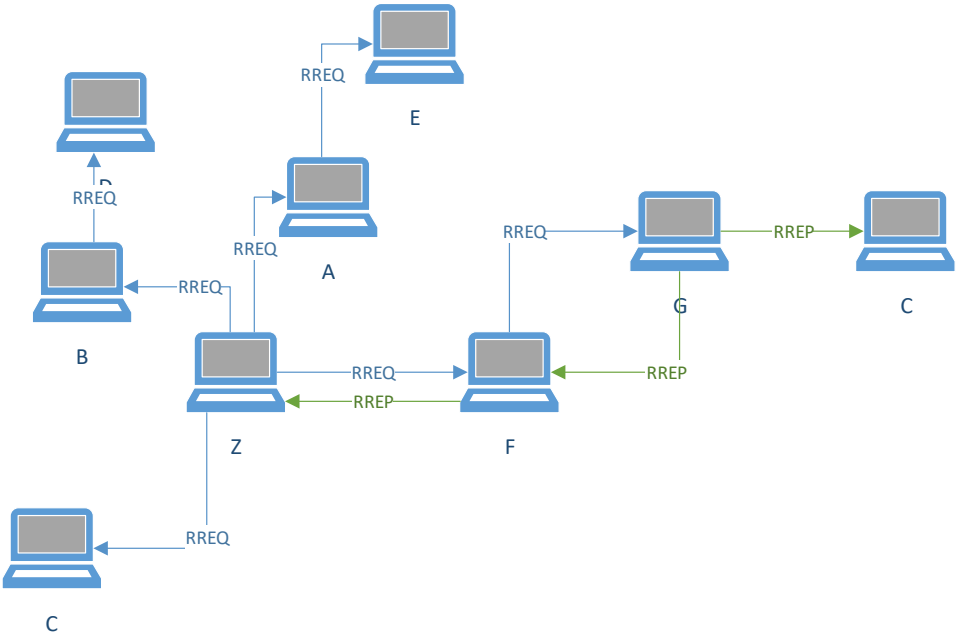
Keď chce zdrojový uzol poslať dátový paket pre cieľový uzol, musí ako prvé vedieť trasu k tomuto uzlu. Zdrojový uzol **Z** nahliadne do svojej smerovacej tabuľky a zistí či nemá záznam o smerovaní k cieľovému uzlu **C**. Ak sa takýto záznam v jeho smerovacej tabuľke nachádza, pošle paket touto trasou k cieľovému uzlu.
V prípade, že uzol **Z** takýto záznam nemá, musí začať proces objavovania trasy za pomoci broadcastového vysielania RREQ správy zobrazenej na obr.3.5. Každý uzol skontroluje zdrojovú adresu a ID požiadavku prijatej RREQ správy. Uzly, ktoré už RREQ správu zaznamenali a táto správa obsahovala rovnakú zdrojovú adresu a ID požiadavku, zamietnu novu RREQ správu. RREQ ID je zvyšované o 1, zakaždým čo zdrojový uzol odošle RREQ správu.[14][15]



Obr. 3.5: Posielanie správ RREQ a RREP medzi susednými uzlami

Ak je susedný uzol ako je možné vidieť na obr.3.5 aj uzlom cieľovým tak si **C** aktualizuje svoju smerovaciu tabuľku a odošle uzlu **Z** RREP správu ako unicast. Ako náhle zdrojový uzol získa správu RREP trasa je platná.

V prípade, že žiadny zo susedných uzlov nie je cieľový, musí proces objavovania pokračovať ďalej. Susedné uzly, ktoré prijali správu RREQ od **Z**, si uložia informácie o trase do svojej smerovacej tabuľky *(Reverse Path Setup)* a pozrú sa do svojich smerovacích tabuliek či nepoznajú cestu k uzlu **C**. Ak túto cestu nepoznajú posielajú ďalej za pomoci broadcastu správu RREQ všetkým svojím susedom. Toto sa opakuje až kým sa nenájde cieľový uzol. Keď sa nájde uzol **G**, ktorý pozná uzol **C**, pošle sa RREP správa uzlu **A** a tak isto cieľový uzol musí byť informovaný zo strany uzlu **G** a tak sa mu pošle správa RREP ako je znázornené na obr.3.6. Ako náhle je uzlu **A** doručená správa RREP, vytvorí sa trasa medzi **A** a **C** a uzly môžu spolu komunikovať.



Obr. 3.6: Posielanie správ RREQ a RREP medzi uzlami v sieti

Počas toho ako je spojenie medzi uzlami vytvorené, uzly, ktoré tvoria trasu spojenia medzi **A** a **C** si medzi sebou pravidelne zasielajú HELLO správy. Tieto HELLO správy sú informačného charakteru a informujú ostatné uzly o tom, že je trasa stále aktívna.[15] Ak jeden z uzlov, neprijme HELLO správu od svojho susedného uzlu v stanovenom čase, ktorý sa volá HELLO interval tak potom:

- Záznam o tomto uzle v tabuľke sa stáva neplatným.
- RERR správa je vygenerovaná a poslaná susedným uzlom. V správe je informácia o tom, že na trase nastalo prerušenie spojenia.

Ak sa počas procesu objavovania trasy zistí zlyhanie ktoréhokoľvek uzlu a je správa RERR vygenerovaná, uvedie sa zneplatnená adresa tohto uzla do zoznamu a odošle sa všetkým ostatným uzlom, ktoré používajú danú cestu ku svojej komunikácii. Ako náhle zdrojový uzol dostane správu RERR musí znova vyžiadať proces objavovania trasy a vyžiadať si tak nové smerovanie.

4 The expected transmission count

ETX(The expected transmission count) metrika je taká, ktorá hľadá cestu s najväčšou priepustnosťou vo viac skokovej bezdrôtovej sieti. Sieť s najmenším ETX má najväčšiu priepustnosť.[16] ETX metrika cesty je predpokladaný počet prenesených dát potrebný k poslaniu paketu cez túto cestu do ktorej je zahrnuté aj preposielanie dát. Paket, ktorý nie je úspešne prijatý bude znova odoslaný odosielateľom dát.
Keď p_f predstavuje pravdepodobnosť úspešného prenesenia paketu a p_r pravdepodobnosť úspešného prijatia ACK paket, pravdepodobnosť úspešného odoslania a následného priajtia paketu bude $p_f \cdot p_r$. ETX pre linku l bude nasledujúci

$$ETX_l = \frac{1}{p_f \cdot p_r}, \tag{4.1}$$

pravdepodobnosti p_f a p_r sú merané pomocou vyhradených snímacích paketov LPP. Každý uzol posiela pomocou broadcastu LPP s rovnakou veľkosťou, za priemernú periódu τ. Pretože tieto snímajúce pakety sú vysielané broadcastom, uzly ich neakceptujú ani nepreposielajú.[17] Každý uzol si pamätá počet LPP, ktoré prijal počas posledný w sekúnd a to umožňuje vypočítanie pravdepodobnosti p_r v hocijakom čase nasledujúc:

$$p_r = \frac{count(t-w, t)}{\frac{w}{\tau}}, \tag{4.2}$$

$count(t-w, t)$ je počet LPP prijatých počas sledovaného okna w a $\frac{w}{\tau}$ je počet LPP, ktoré mali byť prijaté. Hodnota w je určená ako $w = 10\tau$ a pre τ je hodnota $\tau = 1s$.[17] Na vypočítanie liniek ETX vyžaduje obe hodnoty p_f a p_r. Každý LPP poslaný uzlom X obsahuje počet LLP prijatých uzlom X od každého z jeho susediacich uzlov, za posledných w sekúnd. Toto umožňuje každému susediacemu uzlu vypočítanie hodnoty p_f k uzlu X, kedykoľvek susediaci uzol dostane LPP od uzlu X. Metrika trasy r je následne súčet ETX hodnôt každej z liniek v trase[17]:

$$ETX_r = \sum_{l \in r} ETX_l. \tag{4.3}$$

4.1 Implementácia LPP a ETX v AODV

Aby bolo možné použiť v protokole AODV metriku ETX, musí sa najskôr implementovať LPP paket. Každý uzol posiela za pomoci broadcastu LPP pakety každú sekundu a zapamätáva si LPP, ktoré prijme od susedných uzlov za posledných 10 sekúnd. AK chceme počítať ETX metriku, museli byť v každom uzle vytvorené nové susediace tabuľky, ktoré obsahujú nasledujúce tri polia:

- **NeighborIpAddress**: obsahuje IP adresy susediacich uzlov.

- **ReverseLppCount**: zaznamenáva počet LPP paketov prijatých od každého susediaceho uzlu v rámci posledných 10 sekúnd.
- **ForwardLppCount**: hodnota je získaná z prijatých LPP paketov.[17]

Vďaka týmto hodnotám pre každý susedný uzol vie uzol vypočítať ETX metriku pre svojho suseda kedykoľvek.
Aby AODV mohlo fungovať s ETX metrikou, musia byť pridané polia ETX do protokolu. V AODV protokole sú smerovacie tabuľky klasifikované podľa destinácie a to podľa počtu skokov. Ak AODV nachádza viac smerovacích ciest k cieľovému uzlu, potom protokol vyberá za najlepšiu cestu, ktorá obsahuje najmenší počet skokov. Pri upravenom AODV–ETX protokole je cesta klasifikovaná na základe ETX metriky. Najlepšia cesta je tá, ktorá má metriku ETX najmenšiu. Ak sa nájde viacero ciest, ktoré túto metriku majú rovnakú vyberá sa následne tá, ktorá ma najmenší počet skokov k cieľovému uzlu.[17]

ETX metrika je pridaná upravením RREQ a RREP paketov. Do týchto paketov sa pridá ďalšie pole, ktoré reprezentuje ETX hodnotu. Ako náhle zdrojový uzol **Z** vyšle RREQ nastaví sa počiatočná hodnota ETX na nulu. Toto pole je spravované rovnako ako pole počítania skokov. Ako náhle je RREQ správa posielaná ďalej k cieľovému uzlu, ETX metrika sa aktualizuje pre každý uzol na základe rovnice 4.3. Rovnaký postup sa uplatňuje aj pri správe RREP.[17]

ETX pole má veľkosť 32 bitov. Vypočítaná hodnota ETX pomocou rovnice 4.1 je násobená 10^4 a zaokrúhlená na najbližšiu hodnotu *integer*. Vďaka tomuto sa zabezpečí správne fungovanie ETX metriky v Network Simulator 3.[17]

V základnom protokole AODV platí, že uzly neodpovedajú na žiadnu inú RREQ správu od uzlu od ktorého rovnakú správu s rovnakým ID už dostali.
Pri AODV–ETX je to však iné a uzol, ktorý dostane správu, od uzlu od ktorého rovnakú správu už zaznamenal, posiela RREP správu ak hodnota ETX z prijatej RREQ správy je nižšia ako hodnota zaznamenaná v smerovacej tabuľke alebo aj ak je ETX hodnota rovnaká, ale hodnota počtu skokov je nižšia.[17]

5 Network Simulator 3

NS–3(Network Simulator 3) je sieťový simulátor diskrétnych udalostí zameraný predovšetkým na výskum a výuku. Projekt NS–3 začal v roku 2006 ako open-source projekt vyvíjaný NS–3. Bol vytvorený s cieľom poskytnúť otvorenú, rozšíriteľnú platformu pre výskum sietí a vzdelávania. Poskytuje modely fungovania a výkonu paketových dátových sietí a poskytuje simulačné prostriedky pre užívateľov na vykonávanie rôznych simulačných experimentov.
Dôvod na používanie tohto simulátoru je ten, že poskytuje možnosť vykonávať zložitejšie simulácie pre účely vzdelávanie alebo výskumu ako je možné v skutočnosti vykonať s danými systémami.
Je postavený na základoch programovacích jazykov C++ a Python. Tento simulátor je vyvíjaný pre Unixové operačné systémy pod licenciou GNU GPLv2. Podporuje IP aj non-IP siete. Každopádne väčšina užívateľov sa pri používaní zameriava na bezdrôtové/IPsimulácie ako Wi–Fi, WiMAX alebo LTE pre prvú a druhú vrstvu. NS–3 podporuje plánovač v reálnom čase, ktorý napomáha k množstvu prípadov využitia pre „simulácie v slučke“. Pre príklad používatelia môžu vysielať a prijímať pakety generované v NS–3 na skutočných zariadeniach.
Je navrhnutý ako balíček knižníc, ktoré môžu byť spolu kombinované a tiež kombinované s externým softwarom. Neposkytuje grafické rozhranie a tak po ukončení simulácii je potreba použitie nástrojov tretích strán na zobrazenie grafických výstupov. NS–3 sa skladá z piatich typov sieťových komponentov:

- Node (uzol)
- Application (aplikácia)
- Network device (sieťové zariadenia)
- Channel (kanál)
- Topology generator (generátor topológie)

NS–3 nie je oficiálne podporovaným produktom žiadnej firmy. Podporu pre NS–3 zaisťujú používatelia na NS–3 fórach.[18]

6 Inštalácia NS–3 v Ubuntu 18.04

Pre prácu s NS–3 som využil voľne dostupný linuxový systém Ubuntu[1] 18.04 64–bit verziu. Ubuntu som následne inštaloval ako virtuálnu mašinu v programe VMware Workstation 15 Player[2] (distribúcia pre 64–bit systémy Windows), nainštalovaný pre využitie na osobné účely. Pri inštalácii a následnej konfigurácii Ubuntu som zvolil nasledovné parametre:

- Pamäť RAM: 4 GB
- Procesory: 2
- Hard Disk: 20 GB

Ostatné parametre zostali nastavené štandardne ako ich ponúkla inštalácia.
Po spustení virtuálneho stroja Ubuntu som následne otvoril terminál a príkazmi

```
$ sudo su
# apt-get update && apt-get upgrade
```

som spravil aktualizáciu systému. Po úspešnej aktualizácii som násladne stiahol ns–3 zo stránky `https://www.nsnam.org/releases/ns-3-30/` presnejšie verziu ns–allinone–3.30.1. Pre inštaláciu simulačného nástroja sú potrebné rôzne balíčky a knižnice, ktoré nie sú štandardne nainštalované v Ubuntu. Všetky balíčky, ktoré boli inštalované pred samotnou inštaláciou ns–3 môžeme nájsť na officiálnej stránke simulátoru[3] vyberám niektoré z nich:

```
apt-get install gcc g++ python python3
apt-get install tcpdump
apt-get install gdb valgrind
```

Simulačný program, ktorý som stiahol musím najskôr rozbaliť, to urobím pomocou terminálu a príkazu:

```
tar jxvf ns--allinone--3.30.1.tar.bz2
```

Rozbalený súbor som si uložil do zložky **Projekt**. Celá kompilácia, vytváranie programu a simulácii bude prebiehať v terminály a pod užívateľom, teda pod „super užívateľom“ root. V otvorenom terminály som vošiel do priečinka kde je rozbalený simulátor:

```
# cd Projekt/ns-allinone-3.30.1
```

Nasledujúcim krokom je kompilácia programu, tento proces je riadený automaticky kompilačným nástrojom a jeho trvanie je rôzne v závislosti od použitého hardwaru v mojom prípade 16 minút. Kompilácia sa spúšťa nasledujúcim príkazom:

[1] https://www.ubuntu.com/download/desktop
[2] Program bol stiahnutý zo stránky https://www.vmware.com
[3] https://www.nsnam.org/wiki/Installation#Ubuntu.2FDebian.2FMint

```
# ./build.py --enable-examples --enable-test
```

Pri kompilácii v mojom prípade nastala chyba: „**g++ internal compiler error : killed (program cc1plus)**“. Chybu sa mi následne podarilo odstrániť pridaním veľkosti swap pamäte na 2 GB

```
# free
# dd if=/dev/zero of=/var/swap.img bs=1024k count=2000
# mkswap /var/swap.img
# swapon /var/swap.img
# free
# make -f makefile.unix
```

Po znovu spustení kompilácie prebehla úspešne s nasledujúcim výpisom zobrazeným na obr.6.1 nižšie.

```
Waf: Leaving directory `/home/lukas/Projekt/ns-allinone-3.30.1/ns-3.30.1/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (15m42.225s)

Modules built:
antenna                  aodv                     applications
bridge                   buildings                config-store
core                     csma                     csma-layout
dsdv                     dsr                      energy
fd-net-device            flow-monitor             internet
internet-apps            lr-wpan                  lte
mesh                     mobility                 mpi
netanim (no Python)      network                  nix-vector-routing
olsr                     point-to-point           point-to-point-layout
propagation              sixlowpan                spectrum
stats                    tap-bridge               test (no Python)
topology-read            traffic-control          uan
virtual-net-device       visualizer               wave
wifi                     wimax

Modules not built (see ns-3 tutorial for explanation):
brite                    click                    openflow

Leaving directory `./ns-3.30.1'
root@ubuntu:/home/lukas/Projekt/ns-allinone-3.30.1#
```

Obr. 6.1: Výpis terminálu po úspešnej kompilácii

Po úspešnej kompilácii som prešiel do priečinka `# cd Projekt/ns-allinone-3.30.1/ns-3.30.1/` a skontroloval `# ls` či sa v ňom nachádzajú súbory `waf`. Súbor waf je kompilačný nástroj na báze programovacieho jazyka Python.[18] Pomocou neho budú spúšťané všetky aplikácie v simulačnom prostredí ns–3. Tento súbor sa najskôr musí nakonfigurovať nasledujúcim príkazom:

```
# ./waf --build-profile=debug --enable-examples
--enable-tests configure
```

Konfigurácia v mojom prípade prebehla okamžite. Po úspešnej konfigurácii sa v terminály objaví hlásenie **'configure' finished successfully.**
V ďalšom kroku som otestoval ns–3 pomocou skriptu `test.py`.

```
# ./test.py
```

Tento test trvá dlhšiu dobu, ale je dôležitý z toho dôvodu, že otestuje všetky moduly v ns–3 [18]. Prebieha paralelne za pomoci Waf kompilátoru a výpis z neho by mal vyzerať nasledovne:

```
635 of 638 tests passed (635 passed, 3 skipped, 0 failed, 0 crashed, 0 valgrind errors)
```

Ak vo výpise objavíme nejaké failures, crashes alebo valgrind errors naznačuje to tomu, že je problém z kódom alebo nie je kompatibilita medzi nástrojmi a kódom. Spustenie skriptu v ns–3 prembieha nasledovne:

```
# ./waf --run hello-simulator
```

Terminál nám vypíše hlásenie ***Hello Simulator***.

6.1 Inštalácia Wireshark a Code::Block

Inštalácia Wireshark je pomerne jednoduchá stačí nám do terminálu napísať príkaz

```
# apt-get install wireshark
```

Wireshark budeme používať na zobrazovanie súborov *pcap*, ktoré budú zachytávané počas simulácii.
Tak isto ako Wireshark, Code::Block nainštaluje pomocou jediného príkazu v terminály

```
# apt-get install codeblocks
```

Tento program budem využívať pre prácu so súbormi s koncovkou *.cc* v C++.

7 Vytvorenie MANET siete v NS–3

Na vytvorenie MANET siete a pri simuláciách bol použitý voľne šíriteľný zdrojový kód pre AODV, ktorý môžeme nájsť na stránkach NS–3.

Na začiatku kódu sú pridané hlavičkové súbory, moduly. Tieto moduly sa používajú pri behu simulácie.

```
#include <iostream>
#include <cmath>
#include "ns3/aodv-module.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/v4ping-helper.h"
#include "ns3/yans-wifi-helper.h"
```

Pre vytvorenie uzlov v sieti a prístupu k ním je následne použitý `NodeContainer`, a metódou `Create` sú vytvorené uzly.

```
CreateNodes ();
nodes.Create (8);
```

Uzly musia mať definované rozmiestnenie v mriežke. Na rozmiestnenie uzlov použijem `MobilityHelper` a funkciu `SetPositionAllocator`.

```
MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (100.0),
                               "MinY", DoubleValue (100.0),
                               "DeltaX", DoubleValue (50),
                               "DeltaY", DoubleValue (0),
                               "GridWidth", UintegerValue (20),
                               "LayoutType", StringValue ("RowFirst"));
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (nodes);
```

Jednotlivé položky znamenajú:

- **MinX, MinY**: začiatočné súradnice
- **DeltaX, DeltaY**: vzdialenosť medzi uzlami
- **GridWidth**: množstvo uzlov umiestnených do siete
- **LayoutType**: spôsob rozloženia uzlov (RowFirst alebo ColumnFirst)

V tomto príklade sú uzly rozmiestnené do riadku, vzdialenosť medzi nimi je 50 metrov. Pre príklad je použitých 20 uzlov, ktorých mobilita je konštantná, teda sa nehýbu.
Keďže potrebujeme zaistiť aby uzly medzi sebou mohli spolu komunikovať pomocou bezdrôtového kanálu musíme použiť helpery `WifiMacHelper`, `YansWifiPhyHelper`, `YansWifiChannelHelper`. Na jednotlivé uzly potom za pomoci metódy `Install`, priraďujeme parametre ako sú fyzická a linková adresa.

```
WifiMacHelper wifiMac;
wifiMac.SetType ("ns3::AdhocWifiMac");
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
wifiChannel.AddPropagationLoss("ns3::RangePropagationLossModel",
"MaxRange",DoubleValue(range));
wifiPhy.SetChannel (wifiChannel.Create ());
WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
"DataMode", StringValue ("OfdmRate6Mbps"), "RtsCtsThreshold",
UintegerValue (0));
devices = wifi.Install (wifiPhy, wifiMac, nodes);
wifiPhy.EnablePcapAll (std::string ("aodv"));
```

Na záver je zapnuté vytváranie pcap súborov, ktoré si prezrieme vo Wiresharku. Súbory obsahujú komunikáciu všetkých uzlov vo vytvorenej sieti. Ako smerovací protokol je v simulácii použitý protokol AODV. V kóde to vyzerá nasledovne:

```
AodvHelper aodv;
   InternetStackHelper stack;
   stack.SetRoutingHelper (aodv);
   stack.Install (nodes);
   Ipv4AddressHelper address;
   address.SetBase ("192.168.1.0", "255.255.255.0");
   interfaces = address.Assign (devices);
```

Adresný priestor pre vytvorené uzly je vytvorený vďaka `Ipv4AddressHelper`. Vo vytvorenej sieti potrebujeme vytvoriť prevádzku. Na to je nutné vytvoriť aplikáciu, ktorá generuje pakety. V našej simulácii je použitý `V4PingHelper`.

```
V4PingHelper ping (interfaces.GetAddress (7));
ping.SetAttribute ("Verbose", BooleanValue (true));

ApplicationContainer p = ping.Install (nodes.Get (0));
   p.Start (Seconds (0));
   p.Stop (Seconds (60));
```

Pomocou objektu `ApplicationContainer` sa nainštaluje aplikácie na zdrojový uzol. Ďalej sa určí kedy má začať aplikácie fungovať a kedy má skončiť. Simulácia je nastavená na 60 sekúnd a tak isto nastavíme aj ukončenie pingu. Počas simulácie sa uzol 4 presunie preč od ostatných uzlov, aby sme mali možnosť zachytiť v komunikácii medzi uzlami správu RRER. Odstránenie uzlu je dosiahnuté nasledujúcou časťou kódu

```
Ptr<Node> node = nodes.Get (4);
Ptr<MobilityModel> mob = node->GetObject<MobilityModel> ();
Simulator::Schedule (Seconds (20), &MobilityModel::SetPosition,
mob, Vector (1e5, 1e5, 1e5));
```

Kde sa v čase 20 sekúnd uzol 4 presunie mimo ostatné uzly. Simulácia sa spúšťa pomocou `Simulator::Run` a po dokončení úloh vypína za pomoci `Simulator::Stop`.

```
Simulator::Stop (Seconds (60));
Simulator::Run ();
Simulator::Destroy ();
```

Pre spustenie simulácie použijeme nástroj `waf`.

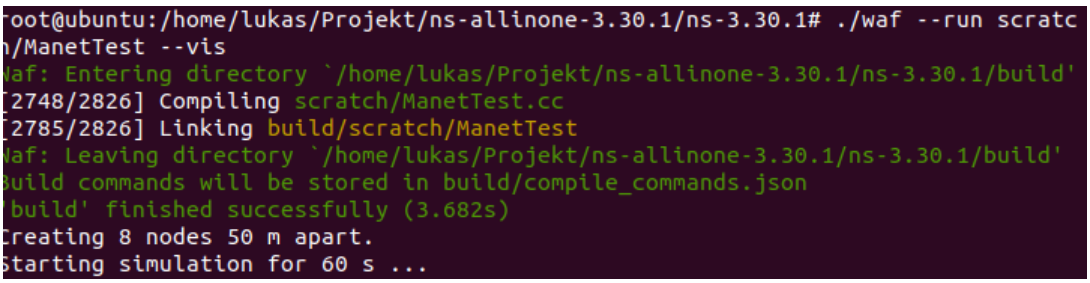
```
# ./waf --run scratch/ManetTest
```

7.1 Simulácia protokolu AODV

Simuláciu si môžeme spustiť aj s grafickým znázornením. Pre spustenie takejto simulácie použijeme príkaz

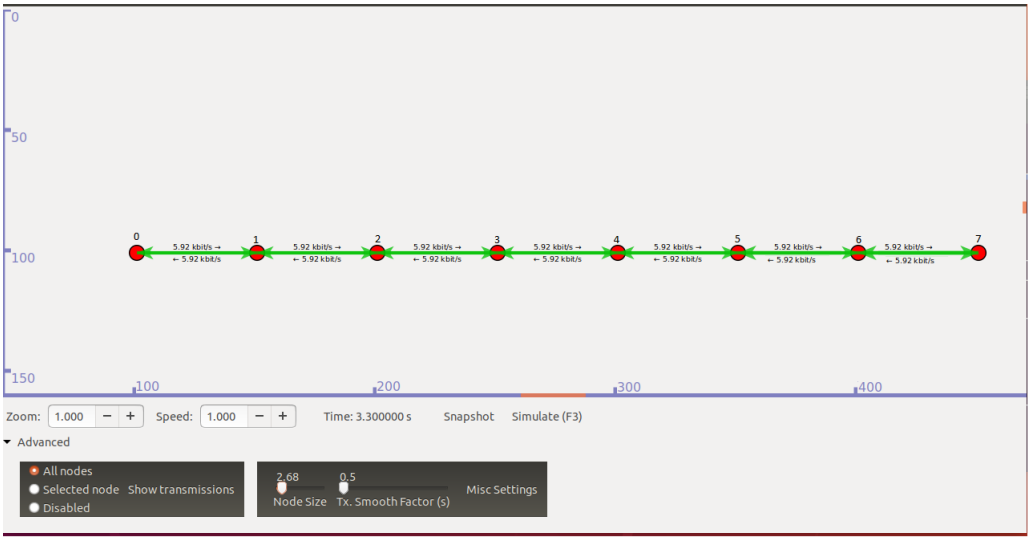
```
# ./waf --run scratch/ManetTest --vis
```

Ako náhle je príkaz zadaný do terminálu objaví sa nám v terminály výpis o tom, že máme spustenú simuláciu o 8 uzloch, ktoré sú od seba vzdialené 50 metrov a táto simulácia potrvá 60 sekúnd. Každý uzol má dosah 60 metrov. Výpis z terminálu môžeme vidieť na obr.7.1.



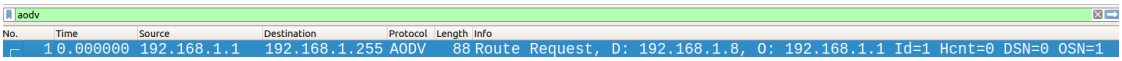
Obr. 7.1: Spustenie simulácie protokolu AODV

Hneď ako začne simulácia, zobrazí sa nám aj grafické znázornenie našej siete obr.7.2 v ktorom môžeme vidieť uzly, ktoré sme vytvorili. V tejto časti simulácie sú všetky uzly v rade, kde v čase 20 sekúnd sa uzol číslo 4 presunie mimo našu topológiu a tak uzol 0, ktorý komunikuje s uzlom 7 stratia spojenie. Pri tejto simulácii nastane strata paketov 66%.Prvých 20 poslaných paketov prejde a ako náhle uzol 4 prestane v topológii existovať, ostatné pakety nie je možné doručiť cieľovému uzlu. Topológia v tejto simulácii je konštantná, to znamená, že uzly sa nehýbu. Po skon-



Obr. 7.2: Grafické zobrazenie simulácie AODV s konštantnou pozíciou

čení 60 sekúnd, je simulácia ukončená a v priečinku *ns-3.30.1* sú uložené súbory pcap, ktoré si môžeme otvoriť v programe Wireshark. Po otvorení Wiresharku si môžeme prezrieť komunikáciu každého uzlu. Po otvorení pcap súborov jednotlivých uzlov som si nastavil filter na protokol AODV.
Uzol 0 si v čase 0 sekúnd požiadal správou RREQ obr.7.3 o zistenie destinácie k cieľu uzlu 7 s cieľovou IP adresou 192.168.1.8. Následne dostal odpoveď RREP od sused-



Obr. 7.3: Route Request zobrazenie vo Wireshark

ného uzlu, ten si požiadal správou RREQ svojho suseda o cestu k uzlu 7 a pomocou mechanizmu spätnej cesty si spravil záznam k uzlu 0. Komunikácia následne pokračovala až k uzlu 6, ktorý v čase 1,189 sekundy obdržal správu RREQ, s cieľovou IP adresou uzlu 7, zdrojovou adresou uzlu 0, počtom skokov 5 a sekvenčným číslo 4 od uzlu 5.

```
Route Request, D: 192.168.1.8, O: 192.168.1.1 Id=4 Hcnt=5 DSN=0 OSN=4
```

Uzol 0 vyslal až 4 RREQ správy, kým bolo naviazané spojenie s uzlom 7. Uzol 6 mal v smerovacej tabuľke zaznamenaný uzol 7 a tak mu poslal správu RREP v čase 1,192 aby informoval uzol 7, že s ním chce uzol 0 nadviazať spojenie.

```
Route Reply, D: 192.168.1.1, O: 192.168.1.8 Hcnt=6 DSN=4 Lifetime=5119
```

Tak isto uzol 6 pošle správu **RREP s príznakom A** susednému uzlu 5, že pozná cestu k uzlu 7.

```
Source       Destination
192.168.1.7  192.168.1.6 AODV
Route Reply, D: 192.168.1.8, O: 192.168.1.1 Hcnt=1 DSN=0 Lifetime=2868

Type: Route Reply (2)
    Flags: 16384 A
    Prefix Size: 0
    Hop Count: 1
    Destination IP: 192.168.1.8
    Destination Sequence Number: 0
    Originator IP: 192.168.1.1
    Lifetime: 2868
```

Uzol 5 tak následne musí poslať uzlu 7 správu o potvrdení **RREP Acknowledgment**. Uzol 0 v čase 1,252 prijme správu RREP od uzlu 1 s sekvenčným číslom 0, a počtom skokov k uzlu 7.

```
Route Reply, D: 192.168.1.8, O: 192.168.1.1 Hcnt=6 DSN=0 Lifetime=2868
```

Týmto je proces zisťovania trasy ukončený a vo vybranej simulácii pri 8 uzloch v rade, kde komunikuje prvý uzol s posledným trval **1,252** sekundy.
Komunikácia medzi uzlami 0 a 7 pokračovala až do 20 sekundy. V čase 20 sekúnd sa uzol číslo 4 odpojil od topológie a medzi uzlami 3 a 5 sa vytvoril priestor, cez ktorý už ďalej komunikácia nemohla prebiehať. Ako náhle uzol 4 nebol súčasťou topológie uzol 5 vyslal RRER správu, ktorej obsah je nasledovný:

```
Ad hoc On-demand Distance Vector Routing Protocol, Route Error
    Type: Route Error (3)
    Flags: 0
    Destination Count: 2
    Unreachable Destinations
        Unreachable Destination IP: 192.168.1.1
        Destination Sequence Number: 4
        Unreachable Destination IP: 192.168.1.5
        Destination Sequence Number: 0
```

Týmto sa komunikácia medzi uzlami 0 a 7 ukončila, pretože simulácia v tomto prípade nemá body, ktoré sa hýbu a tak komunikácia ďalej prebiehať nemohla. Všetky ostatné správy od uzlu 0 boli zahodené. Pre porovnanie boli zhotovené ďalšia simulácie, zobrazené v tab.7.1. Všetky simulácie začínajú od času 0 s, po odoslaní prvej správy RREQ od strany zdroja k poslednému uzlu v topológii.

Počet uzlov	***Prvý RREQ [s]***	***RREP[s]***
3	0	0,254
8	0	1,242
15	0	2,028
20	0	2,129
30	0	2,233

Tab. 7.1: Porovnanie času potrebného na objavenie cesty AODV protokolom

7.1.1 Simulácia s usporiadaním RandomRectanglePositionAllocator

V tomto druhu simulácie sú uzly v našej sieti vytvorené na náhodných pozíciach. Dosiahneme to tak, že použijeme triedu `ObjectFactory`.

```
  ObjectFactory pos;
  pos.SetTypeId ("ns3::RandomRectanglePositionAllocator");
  pos.Set ("X", StringValue ("ns3::UniformRandomVariable
[Min=0.0|Max=100.0]"));
  pos.Set ("Y", StringValue ("ns3::UniformRandomVariable
[Min=0.0|Max=100.0]"));
  Ptr<PositionAllocator> taPositionAlloc =
  pos.Create ()->GetObject<PositionAllocator> ();
  streamIndex += taPositionAlloc->AssignStreams (9);
```

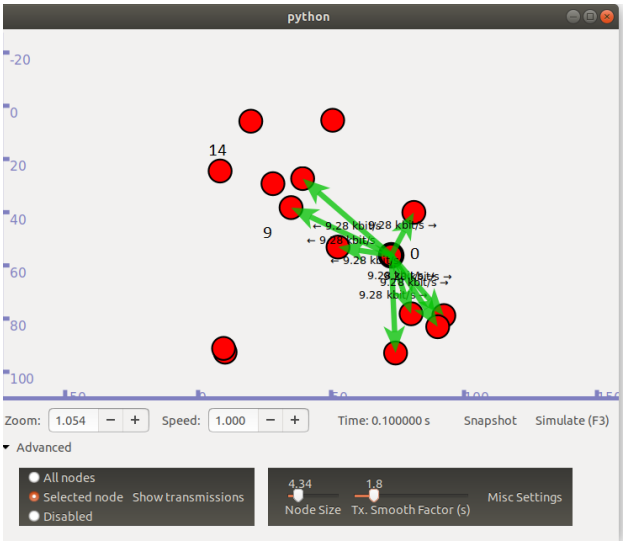
Všetky popisované simulácie sú založené na komunikačnom scenári kde uzol 0 (prvý uzol vytvorenej topológie) komunikuje s posledným uzlom topológie. Pri sieti o 15 uzloch sú to uzly 0 a 14.
Parametre simulácie:

- Priestor simulácie: 100 metrov na 100 metrov
- Počet uzlov: 15
- Trvanie simulácie: 60 sekúnd
- Vzdialenosť uzlov od seba: náhodná
- Dosah uzlov: 60 metrov

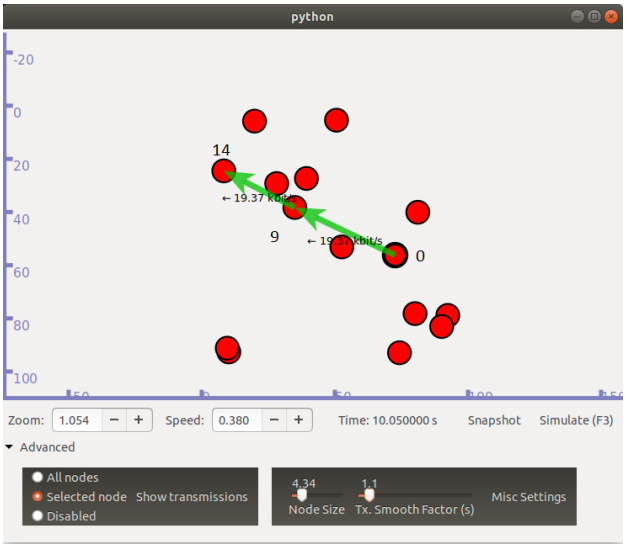
Po zadaní príkazu do terminálu na zahájenie simulácie

```
# ./waf --run scratch/AODV_random_static --vis
```

sa nám uzly usporiadajú náhodne a hneď po spustení simulácie uzol 0 začne s procesom objavovania trasy zobrazenom na obr.7.4. Akonáhle nájde trasu k uzlu 14, začnú spolu komunikovať. Celá komunikácia prebieha cez susedný uzol 9 ako je možné vidieť na obr.7.5.



Obr. 7.4: Proces objavovania trasy v simulácii

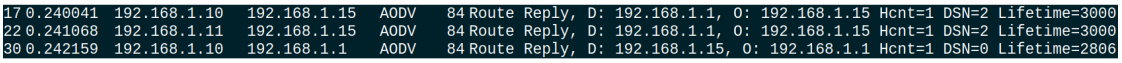


Obr. 7.5: Komunikácia uzlov v grafickom prostredí

Každý uzol si vedie vlastnú smerovaciu tabuľku, ktorá obsahuje IP adresu cieľa, cestu cez ktorý susedný uzol sa tam dostane, rozhranie, značku či je cesta aktuálna, dobu uplynutia platnosti trasy a počet skokov od cieľa. Zo simulácie som vybral časti z troch smerovacích tabuliek uzlov 0, 9 a 14 cez ktoré prebieha komunikácia.

```
Node: 0; Time: +8.0s, Local time: +8.0s, AODV Routing table

Uzol 0 IP address 192.168.1.1

AODV Routing table
Destination     Gateway       Interface Flag  Expire         Hops
192.168.1.10  192.168.1.10  192.168.1.1 UP    2.05             1
192.168.1.15  192.168.1.10  192.168.1.1 UP    2.00             2
192.168.1.255 192.168.1.255 192.168.1.1 UP    9223372028.85    1


Node: 9; Time: +8.00s, Local time: +8.00s, AODV Routing table

Uzol 9 IP address 192.168.1.10

AODV Routing table
Destination     Gateway       Interface Flag  Expire         Hops
192.168.1.1   192.168.1.1   192.168.1.10  UP  2.24             1
192.168.1.15  192.168.1.15  192.168.1.10  UP  2.06             1
192.168.1.255 192.168.1.255 192.168.1.10  UP  9223372028.85    1

Node: 14; Time: +8.00s, Local time: +8.00s, AODV Routing table

Uzol 14 IP address 192.168.1.15

AODV Routing table
Destination     Gateway       Interface Flag  Expire         Hops
192.168.1.1   192.168.1.10  192.168.1.15  UP      2.00         2
192.168.1.10  192.168.1.10  192.168.1.15  UP      2.05         1
192.168.1.255 192.168.1.255 192.168.1.15  UP  9223372028.85    1
```

V programe Wireshark obr.7.6 je vidieť, že uzol číslo 9 ako prvý reagoval na výzvu uzlu 0 a poslal RREP, s počtom skokov 1 a tak uzol 0 naviazal spojenie práve cez tento uzol. Uzol 10, čo je susedný uzol a má tiež počet skokov 1 poslal správu RREP o 0,001s neskôr.



Obr. 7.6: RREP správy Wireshark

7.1.2 Simulácia s usporiadaním RandomRectanglePositionAllocator a náhodným pohybom RandomWaypointMobilityModel

Najbežnejšia situácia s akou sa môžeme v MANET stretnúť je mobilita uzlov. Každý účastnícky uzol v sieti sa pohybuje a spolu s tým sa musí meniť aj smerovanie. K návrhu takejto simulácie nám pomôže `RandomWaypointMobilityModel`.

```
mobility.SetMobilityModel ("ns3::RandomWaypointMobilityModel",
                    "Speed", StringValue (ssSpeed.str ()),
                    "Pause", StringValue (ssPause.str ()),
                    "PositionAllocator", PointerValue (taPositionAlloc));
```

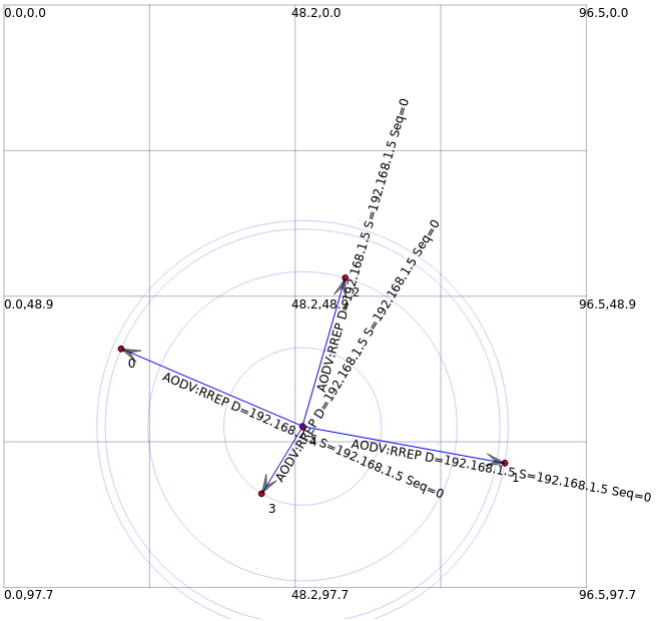
Parametre začiatočnej simulácie:

- Priestor simulácie: 100 metrov na 100 metrov
- Počet uzlov: 5
- Trvanie simulácie: 60 sekúnd
- Vzdialenosť uzlov od seba: náhodná
- Rýchlosť uzlov: 25 m/s
- Dosah uzlov: 60 metrov

V tejto simulácii budeme následne používať vizualizačný nástroj **NetAnim**. Nástroj je súčasťou NS–3 a pre jeho použitie je nutné pridať do zdrojového kódu nasledovné riadky:

```
#include "ns3/netanim-module.h"
AnimationInterface anim ("AODV_random_pohyb.xml");
anim.EnablePacketMetadata(true);
```

Po skončení simulácie si môžeme všimnúť, že pri počte 5 uzlov a veľkosti priestoru 100 metrov na 100 metrov, máme stratu paketov 10%. Zobrazenie simulácie v grafickom prostredí NetAnim si môžete pozrieť na obr.7.7.



Obr. 7.7: Komunikácia uzlov v grafickom prostredí NetAnim

Výsledky z ďalších simulácii sú zobrazené v tab.7.2. Z výsledkov simulácii vyplýva, že pri pridávaní uzlov do siete sa nám zvyšuje stratovosť a oneskorenie paketov. Pri pohybe uzlov sa často krát linka medzi zdrojom a cieľom preruší a tak musí nastať znova proces objavenia a tým sa nám zvyšuje aj oneskorenie liniek.

Počet uzlov	***Stratovosť paketov [%]***	***RTT (round-trip time) avg [ms]***
10	6	2278
15	8	328
25	10	768
35	38	1090
40	43	1818

Tab. 7.2: Porovnanie stratovosti paketov a round-trip time pri zvyšovaní počtu uzlov

7.2 Porovnanie protokolov AODV a AODV–ETX

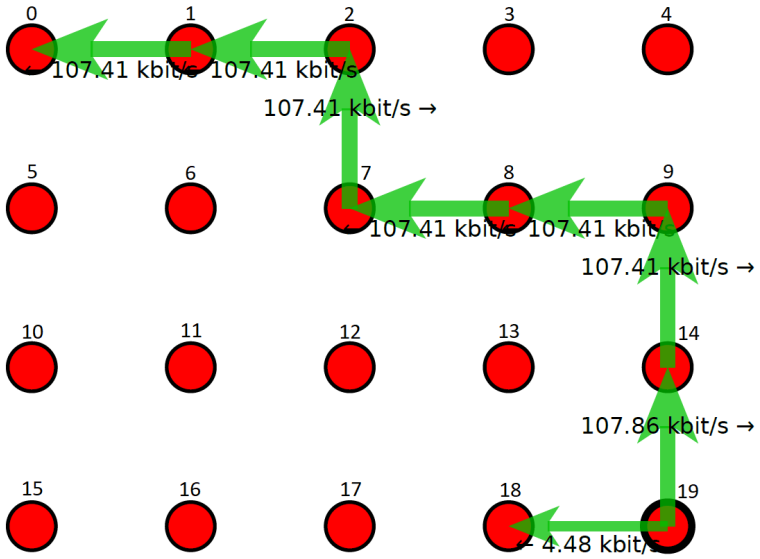
Pri simulácii protokolov ADOV a AODV–ETX boli použité dva simulačné scenáre.

7.2.1 Simulácia pri konštantnom usporiadaní uzlov

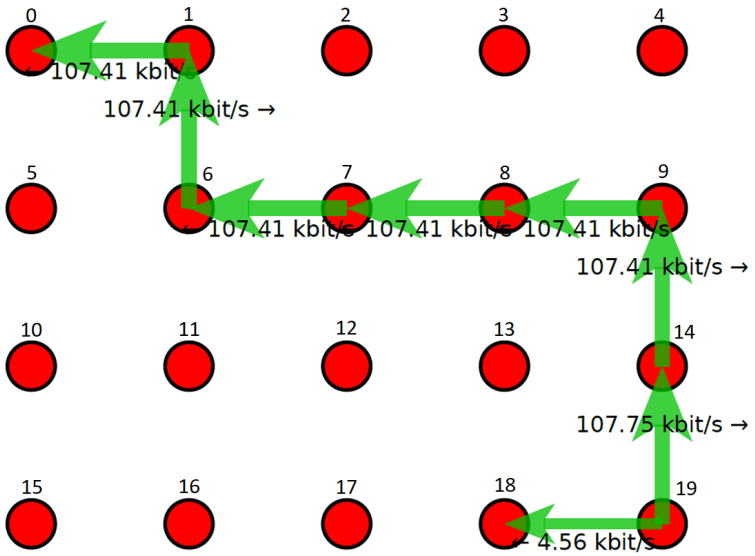
Pri simulácii boli použité rovnaké zdrojové kódy, ktoré využívajú **ConstantPositionMobilityModel**. V modeli komunikujú uzol **0 klient** s uzlom **19 serverom**. Parametre začiatočnej simulácie:

- Protokol používaný pri komunikácii: ns3::UdpSocketFactory
- Počet uzlov v simulácii: 20
- Trvanie simulácie: 20 sekúnd
- Vzdialenosť uzlov od seba: 50 metrov
- Dosah uzlov: 100 metrov

Oba protokoly si vybrali trasu s počtom skokov 7. Výbery týchto trás sú zobrazené na obrázkoch nižšie. AODV protokol si vybral trasu, ktorá je vyobrazená na obr.7.8 a AODV–ETX protokol si vybral trasu, ktorá je vyobrazená na obr.7.9



Obr. 7.8: Výber trasy AODV protokol



Obr. 7.9: Výber trasy AODV–ETX protokol

Zo všetkých testovaní nám vyšli nasledujúce výsledky zobrazené v tab.7.3. AODV a AODV–ETX protokoly sa pri konštantom usporiadaní uzlov nelíšia. Uzly majú podobné cesty, rovnaký počet skokov a aj rovnakú stratovosť a tak sú výsledky simulácii skoro identické. Pre testovanie odlišností týchto dvoch protokolov využijeme aj RandomWalk2dMobilityModel, kde výsledky budú lepšie reprezentovať rozdielnosť protokolov.

Prenosová rýchlosť[kbps]	**Protokol**	**Počet skokov**	**Priepustnosť [kbps]**	**Strata paketov[%]**	**Priemerné oneskorenie paketov[ms]**
100	***AODV–ETX***	7	95,994	6,99	16,250
100	***AODV***	7	95,993	6,99	16,330
120	***AODV–ETX***	7	115,096	7,03	14,684
120	***AODV***	7	115,097	7,03	14,722
140	***AODV–ETX***	7	133,950	7,23	13,569
140	***AODV***	7	133,950	7,23	13,581
160	***AODV–ETX***	7	153,052	7,23	12,694
160	***AODV***	7	153,054	7,23	12,710

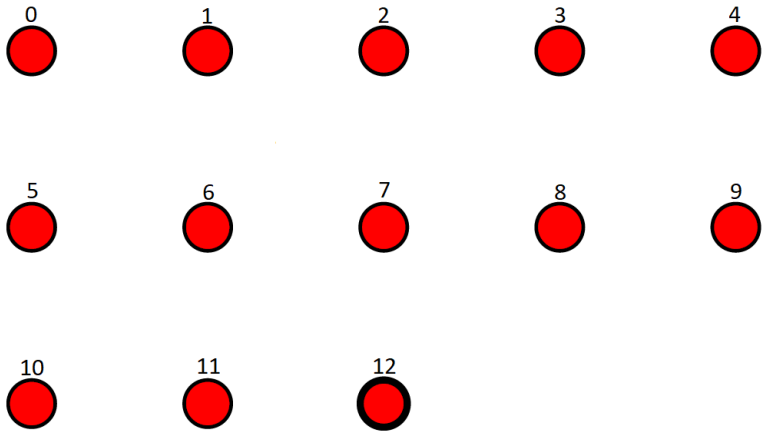
Tab. 7.3: Výsledky simulácie pri konštantnom usporiadaní uzlov.

7.2.2 Simulácia pri použití RandomWalk2dMobilityModel

Tento druh simulácie spočíva v rôznom pohybe uzlov v rámci našej vytvorenej siete. V simulácii sú vždy *dva statické uzly* a to uzol 0 klient a posledný uzol v simulácii server. Pri simulácia boli využité následné parametre:

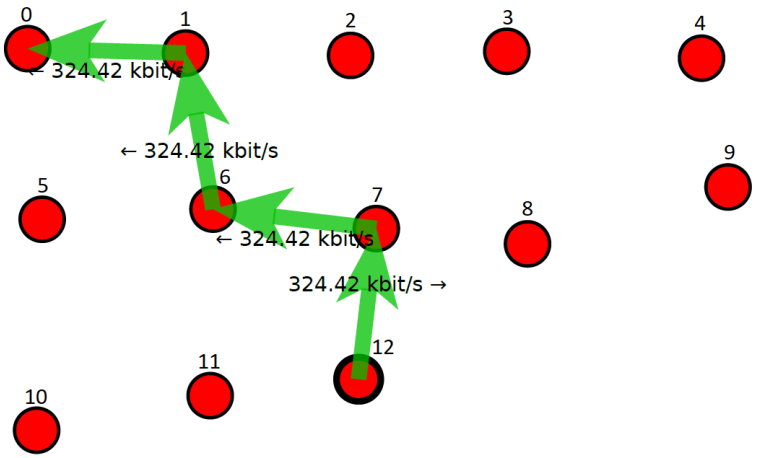
- Protokol používaný pri komunikácii: ns3::UdpSocketFactory
- Počet uzlov v simulácii: rôzny
- Prenosová rýchlosť: 300 kbps
- Trvanie simulácie: 20 sekúnd
- Vzdialenosť uzlov od seba: 50 metrov
- Dosah uzlov: 80 metrov
- Rýchlosť pohybu uzlov: 1,5 m/s

Na začiatku každej simulácie sa uzly rozvrhnú do mriežky, podľa pravidla 5 uzlov na riadok. Pri simulácii 13 uzlov sieť vyzerá nasledovne obr.7.10.



Obr. 7.10: Začiatok simulácie AODV–ETX protokol

Po spustení simulácie začne uzol 12 v čase 1 sekunda komunikovať s uzlom 0. Ako náhle je simulácia spustená všetky uzly okrem uzlov 0 a 12 sa začnú pohybovať konštantou rýchlosťou. Komunikácia a rozloženie uzlov po 13 sekundách od spustenia simulácie vyzerá nasledovne obr obr.7.11.



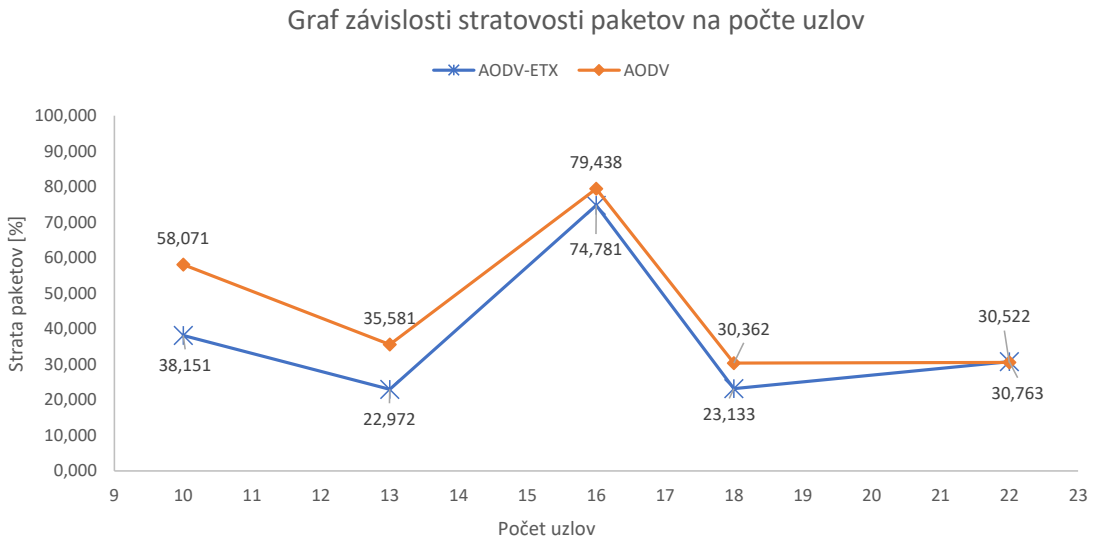
Obr. 7.11: Po 13 sekundách od začiatku simulácie AODV–ETX protokol

Simulácie boli uskutočnené pre protokol AODV a protokol AODV–ETX s rovnakými parametrami ako aj s rovnakou inštaláciou linuxového systému Ubutnu a zároveň aj simulačným prostredím NS3 verzie 3.30.1. Výsledky simulácii pri zmene premennej počtu uzlov sú popísané v tab.7.4.

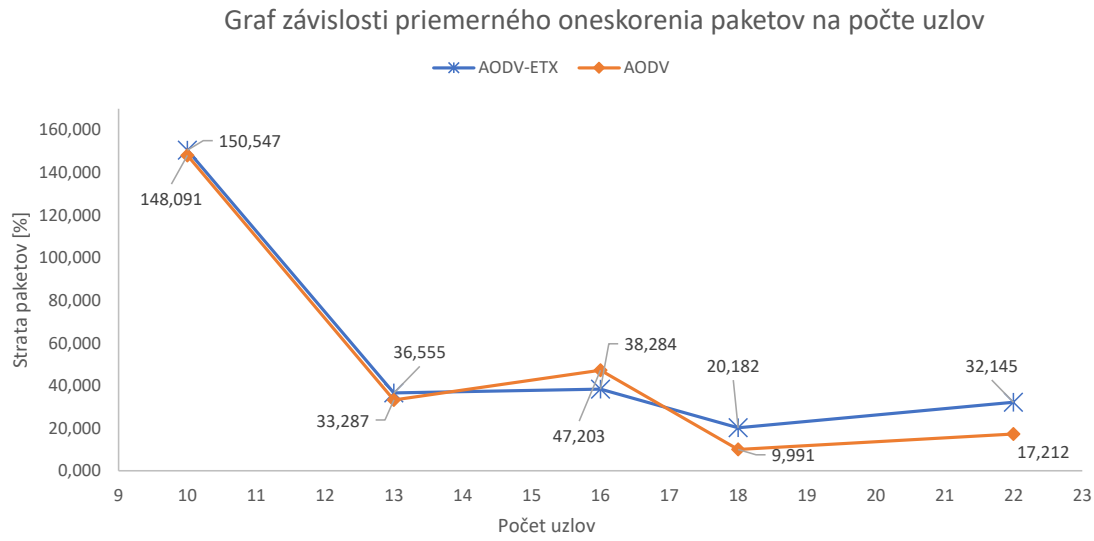
Počet uzlov	**Protokol**	**Priepustnosť [kbps]**	**Strata paketov[%]**	**Priemerné oneskorenie paketov [ms]**
10	***AODV–ETX***	214,38	38,150	150,547
10	***AODV***	134,35	58,070	148,091
13	***AODV–ETX***	227,22	22,972	36,555
13	***AODV***	189,97	35,580	33,287
16	***AODV–ETX***	157,74	74,780	38,284
16	***AODV***	134,94	79,438	47,203
18	***AODV–ETX***	201,86	23,133	20,182
18	***AODV***	198,38	30,362	9,990
22	**AODV–ETX**	171,27	30,763	32,145
22	**AODV**	176,74	30,522	17,212

Tab. 7.4: Výsledky simulácie pri použití RandomWalk2dMobilityModel

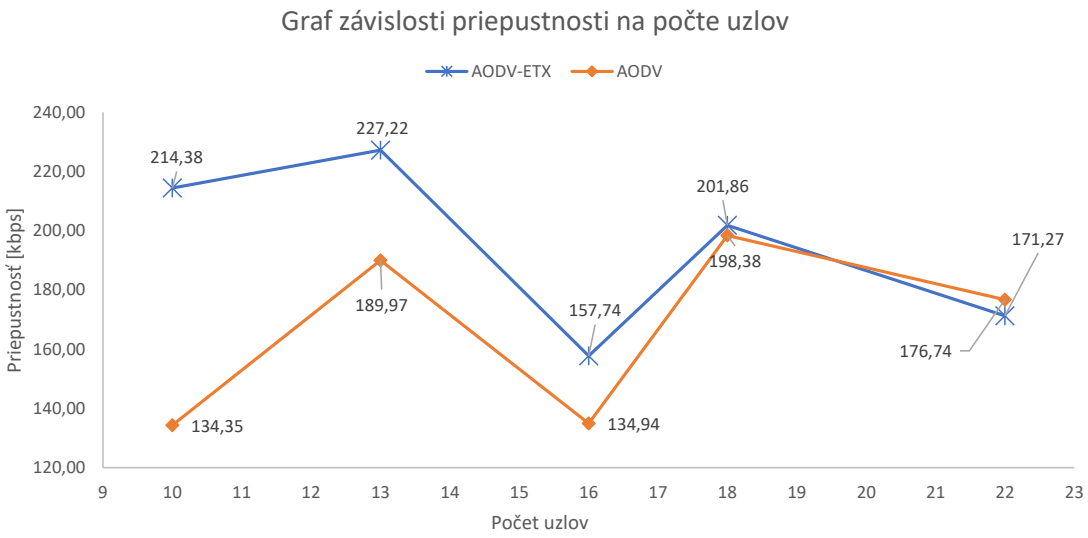
Z výsledkov vyplýva, že protokol AODV–ETX má lepšie vlastnosti pri zostavení komunikácie v prostredí kde sa uzly pohybujú. Môžeme si všimnúť, že pri zväčšení počtu uzlov nad 20 už protokol AODV–ETX nemá také dobré výsledky ako pri menších počtoch uzlov v sieti. Výsledky zobrazované v tab.7.4 boli vynesené do grafov 7.12, 7.13 a 7.14.



Obr. 7.12: Graf závislosti stratovosti paketov na počte uzlov



Obr. 7.13: Graf závislosti priemerného oneskorenia paketov na počte uzlov



Obr. 7.14: Graf závislosti priepustnosti na počte uzlov

Záver

Cieľom tejto diplomovej práce bolo naštudovanie siete MANET jej protokolov a hlavne protokolu AODV, voľby peeru v AODV protokole a následné vylepšenie tohto mechanizmu.
V teoretickej časti tejto práce bola popísaná sieť MANET. Sú tu rozoberané jej vlastnosti, smerovanie a zároveň aj smerovacie protokoly. V práci je ďalej opísané rozdelenie týchto smerovacích protokolov na proaktívne, reaktívne a hybridné. Každá skupina smerovacích protokolov je popísaná a sú k nej priradené konkrétne druhy smerovacích protokolov. Každý smerovací protokol je popísaný a nakoniec je vytvorená súhrna tabuľka vlastností týchto skupín protokolov.

Práca sa ďalej venuje hlavnému protokolu AODV, ktorý je súčasťou praktickej časti tejto práce. V protokole sú rozoberané jeho správy, ich štruktúra a proces objavovania trasy. V práci je popísaná ETX metrika, ktorá bude pridaná k AODV protokolu.

V praktickej časti práce je opísaný Network Simulator 3, s ktorým sa pracuje počas celej práce, ďalej je tu opísaná inštalácia Ubuntu a inštalácia NS–3. V ďalšej kapitole je popísané vytvorenie MANET siete v NS–3, sú tu pridané riadky z kódu, ktorý sa používa pri všetkých simuláciách. V práci sa nachádzajú príkazy na spustenie simulácii, výpisy z terminálu a grafické znázornenia týchto simulácii. Každá simulácia je popísaná a sú tu uplatňované teoretické znalosti zo začiatku práce.

Na konci práce sú vytvorené simulácie protokolu AODV a protokolu AODV–ETX. Voľba peeru pri protokole AODV sa spolieha len na počet skokov, zatiaľ čo pri protokole AODV–ETX sa uplatňuje metrika ETX na zaistenie trasy s najlepšou priepustnosťou. Výsledky simulácii sú zobrazené v tabuľkách a vynesené do grafov.

AODV–ETX prokol s porovnaním k protokolu AODV má lepšie vlastnosti avšak pri väčšom počte uzlov nad 20 sa tieto vlastnosti približujú k protokolu AODV. Pri ďalšom rozšírení práce treba brať v úvahu stratovosti liniek, spraviť ďalšie simulácie aj pri väčšom počte uzlov a rôznych stratovostiach liniek.

Literatúra

[1] MO, Yijun, Jing HUANG a Benxiong HUANG.: *Manet Node Based Mobile Gateway with Unspecific Manet Routing Protocol.* 2006 International Symposium on Communications and Information Technologies [online]. IEEE, 2006, 2006, s. 886-889 [cit. 2019-12-03].
DOI: 10.1109/ISCIT.2006.339864. ISBN 0-7803-9740-1. Dostupné z URL: <http://ieeexplore.ieee.org/document/4141343/>.

[2] Anushka Khattri: *Introduction of Mobile Ad hoc Network (MANET).* [online]. Geeksforgeeks [cit. 2019-12-03]. Dostupné z URL: <https://www.geeksforgeeks.org/introduction-of-mobile-ad-hoc-network-manet/>.

[3] UKEssays.com: *The Characteristics And Applications Of Manets Computer Science Essay.* November 2018 [online].[cit. 2019-12-03]. Dostupné z URL: <https://www.ukessays.com/essays/computer-science/the-characteristics-and-applications-of-manets-computer-science-essay.php?vref=1>.

[4] VADKERTI, G.: *Vytvoření experimentální MANET sítě v simulátoru NS-3.* Brno, 2013. Diplomová práce. FEKT VUT Brno.

[5] UKEssays.com: *Proactive and Reactive MANET Protocols Analysis.* November 2018 [online].[cit. 2019-12-05]. Dostupné z URL: <https://www.ukessays.com/dissertation/examples/computer-science/ad-hoc-network.php?vref=1>.

[6] Study CCNA: *Routing protocols.* Januar 2016 [online].[cit. 2019-12-05]. Dostupné z URL: <https://study-ccna.com/routing-protocols/>.

[7] UKEssays.com: *Mobile Ad Hoc Networks MANET.* November 2018 [online]. [cit. 2019-12-05]. Dostupné z URL: <https://www.ukessays.com/essays/computer-science/advanced-routing-protocols-for-ad-hoc-networks.php>.

[8] Rahman, Abdul Hadi Abd, and Zuriati Ahmad Zukarnain.: *Performance comparison of AODV, DSDV and I-DSDV routing protocols in mobile ad hoc networks.* European Journal of Scientific Research 31.4 (2009): 566-576.[online]. [cit. 2019-12-06]. Dostupné z URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.468.8851&rep=rep1&type=pdf>.

[9] UKEssays.com: *Advanced Routing Protocols for Ad-hoc Networks.* September 2019 [online].Geeksforgeeks [cit. 2019-12-06]. Dostupné z URL: <https://www.ukessays.com/essays/computer-science/advanced-routing-protocols-for-ad-hoc-networks.php?vref=1>.

[10] Thomas Clausen, Philippe Jacquet, C´edric Adjih, Anis Laouiti, Pascale Minet, et al.: *Optimized Link State Routing Protocol (OLSR).* in 2003 Network Working Group. [online]. [cit. 2019-12-06]. Dostupné z URL: <https://hal.inria.fr/inria-00471712>.

[11] MISRA, Padmini.: *Routing protocols for ad hoc mobile wireless networks.* in 1999. Courses Notes. [online]. [cit. 2019-12-06]. Dostupné z URL: <http://suraj.lums.edu.pk/~cs678/papers/17_routing_for_ad_hoc.pdf>.

[12] JOHNSON D., HU Y., MALTZ D.: *The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4.* in February 2007. IETF Network Working Group, RFC 4728. [cit. 2019-12-06]. Dostupné z URL: <https://www.ietf.org/rfc/rfc4728.txt>.

[13] Raman.: *MANET Routing Protocols.* [online].Geeksforgeeks [cit. 2019-12-05]. Dostupné z URL: <https://www.geeksforgeeks.org/manet-routing-protocols/>.

[14] Perkins, Charles, Elizabeth Belding-Royer, and Samir Das.: *RFC 3561: Ad hoc on–demand distance vector (AODV) routing.* [online].Internet RFCs (2003): 1-38 [cit. 2019-12-07]. Dostupné z URL: <https://tools.ietf.org/html/rfc3561>.

[15] Asma Ahmed, A. Hanan, Izzeldin Osman.: *AODV ROUTING PROTOCOL WORKING PROCESS.* [online]. In: Journal of Convergence Information Technology (JCIT). 2015, s. 1-8. [cit. 2019-12-08]. Dostupné z URL: <http://www.globalcis.org/jcit/ppl/JCIT4287PPL.pdf>.

[16] V. Dubey and N. Dubey,: *Performance Evaluation of AODV and AODVETX.* [online]. In: 2014 International Conference on Computational Intelligence and Communication Networks, Bhopal, 2014, pp. 482-485, doi: 10.1109/CICN.2014.112. [cit. 2020-05-30]. Dostupné z URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7065531>.

[17] N. J. Jevtic and M. Z. Malnar,: *The NS-3 simulator implementation of ETX metric within AODV protocol*[online]. In: 2017 25th Telecommunication Forum

(TELFOR), Belgrade, 2017, pp. 1-4, doi: 10.1109/TELFOR.2017.8249315.[cit. 2020-05-30]. Dostupné z URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8249315>.

[18] Ns–3 Tutorial.: *Ns–3 Tutorial: Release ns–3.30* 2019, s. 135 [online]. [cit. 2019-12-08]. Dostupné z URL: <https://www.nsnam.org/docs/release/3.30/tutorial/ns-3-tutorial.pdf>.

Zoznam symbolov, veličín a skratiek

AODV	Ad-Hoc On Demand Distance Vector
CGSR	Clusterhead Gateway Switch Routing
CPU	Centrálna Procesorová Jednotka – Central Processing Unit
DSDV	Destination Sequenced Distance Vector
DSR	Dynamic Source Routing protocol
ETX	The expected transmission count
IP	Internet Protocol
LCC	Least Cluster Change
MANET	Mobilná Ad–hoc sieť – Mobile Ad hoc Network
MPR	Multipoint relays
NS–3	Network Simulator 3
OSLR	Optimized Link State Routing
RPS	Nastavenie spätnej cesty – Reverse Path Setup
RREP	Odpoveď na trasu – Route Reply
RREP–ACK	Potvrdenie odpovede na trasu – Route Reply Acknowledge
RERR	Chyba na trase – Route Error
RREQ	Požiadavka na trasu – Route Request
TORA	Temporally Ordered Routing Algorithm
UDP	Používateľský Datagramový Protokol – User Datagram Protocol
ZRP	Zone Routing Protocol